Programmer's Reference Manual Volume 1

© 1983-1993 The Santa Cruz Operation, Inc.
© 1980-1990 Microsoft Corporation.
© 1989-1990 UNIX System Laboratories, Inc.
All Rights Reserved.

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of the copyright owner, The Santa Cruz Operation, Inc., 400 Encinal, Santa Cruz, California, 95060, U.S.A. Copyright infringement is a serious matter under the United States and foreign Copyright Laws.

The copyrighted software that accompanies this manual is licensed to the End User only for use in strict accordance with the End User License Agreement, which should be read carefully before commencing use of the software. Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc.

SCO OPEN DESKTOP Software is commercial computer software and, together with any related documentation, is subject to the restrictions on U.S. Government use as set forth below.

If this procurement is for a DOD agency, the following DFAR Restricted Rights Legend applies:

RESTRICTED RIGHTS LEGEND:
USE, DUPLICATION OR DISCLOSURE BY THE GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN SUBPARAGRAPH (c)(1)(ii) OF RIGHTS IN TECHNICAL DATA AND COMPUTER SOFTWARE CLAUSE AT DFARS 252.227-7013. CONTRACTOR/MANUFACTURER IS THE SANTA CRUZ OPERATION, INC., 400 ENCINAL STREET, SANTA CRUZ, CA 95060.

If this procurement is for a civilian government agency, the following FAR Restricted Rights Legend applies:

RESTRICTED RIGHTS LEGEND:
THIS COMPUTER SOFTWARE IS SUBMITTED WITH RESTRICTED RIGHTS UNDER GOVERNMENT CONTRACT NO._____ (AND SUBCONTRACT NO._____, IF APPROPRIATE). IT MAY NOT BE USED, REPRODUCED, OR DISCLOSED BY THE GOVERNMENT EXCEPT AS PROVIDED IN PARAGRAPH (g)(3)(i) OF FAR CLAUSE 52.227-14 OR AS OTHERWISE EXPRESSLY STATED IN THE CONTRACT. CONTRACTOR/MANUFACTURER IS THE SANTA CRUZ OPERATION, INC., 400 ENCINAL STREET, SANTA CRUZ, CA 95060.

SCO, Open Desktop, The Santa Cruz Operation, the Open Desktop logo, and the SCO logo are registered trademarks of The Santa Cruz Operation, Inc. in the USA and other countries.

SCO NFS was developed by Lachman Technology, Inc. based on LACHMAN SYSTEM V NFS.5. LACHMAN is a trademark of Lachman Technology, Inc. NFS is a registered trademark of Sun Microsystems, Inc.

SCO TCP/IP was developed by Lachman Technology, Inc. **SCO TCP/IP** is derived from LACHMAN SYSTEM V STREAMS TCP, a joint development of Lachman Technology, Inc. and Convergent Technologies. LACHMAN is a trademark of Lachman Technology, Inc.

All other brand and product names are or may be trademarks of, and are used to identify products or services of, their respective owners.

Date: 6 January 1993
Document Version: 3.2.4E

Preface 1

Documentation conventions 2
Development System documentation set 3
Reference manual sections 7
Alphabetized list 9

Commands for Programming (CP) 33

Intro(CP) 35
adb(CP) 37
admin(CP) 45
ar(CP) 50
as(CP) 53
cb(CP) 55
cc(CP) 56
cdc(CP) 71
cflow(CP) 73
chkshlib(CP) 76
codeview(CP) 78
comb(CP) 94
conv(CP) 96
convert(CP) 98
cpp(CP) 99
cprs(CP) 103
cscope(CP) 104
ctrace(CP) 108
cxref(CP) 114
dbXtra(CP) 117
dbxtra(CP) 130
delta(CP) 149
dis(CP) 152
dosld(CP) 154
dump(CP) 156
dumpmsg(CP) 158
findstr(CP) 160
gencat(CP) 161
gencc(CP) 162
get(CP) 164
help(CP) 172

i286emul(CP) 173
iconv(CP) 175
insertmsg(CP) 177
ld(CP) 179
lex(CP) 186
lint(CP) 190
list(CP) 194
lorder(CP) 196
lprof(CP) 197
m4(CP) 200
make(CP) 205
mar(CP) 214
masm(CP) 216
mcs(CP) 220
mkshlib(CP) 222
mkstr(CP) 227
nm(CP) 229
os2ld(CP) 232
prof(CP) 234
prs(CP) 237
rcc(CP) 241
rcflow(CP) 245
rcxref(CP) 248
regcmp(CP) 250
rlint(CP) 251
rmdel(CP) 255
sact(CP) 256
sccsdiff(CP) 257
sdb(CP) 258
size(CP) 266
strip(CP) 268
tsort(CP) 270
unget(CP) 271
val(CP) 272
vc(CP) 274
what(CP) 277
x286emul(CP) 279
xstr(CP) 280
xtil(CP) 282
yacc(CP) 286

Files for Programming (FP) 289

Intro(FP) 291
86rel(FP) 293
a.out(FP) 298
acct(FP) 304
ar(FP) 305
core(FP) 307
dir(FP) 314
dirent(FP) 315
filehdr(FP) 316
filesystem(FP) 318
inode(FP) 321
langinfo(FP) 322
lconv(FP) 324
ldfcn(FP) 326
limits(FP) 329
linenum(FP) 331
mallinfo(FP) 332
mem(FP) 333
mfsys(FP) 334
msg(FP) 335
netbuf(FP) 337
nl_types(FP) 338
nssend(FP) 339
null(FP) 340
passwd(FP) 341
plot(FP) 343
pnch(FP) 345
proc(FP) 346
reloc(FP) 347
sccsfile(FP) 348
scnhdr(FP) 351
scr_dump(FP) 352
sem(FP) 353
sfsys(FP) 356
shm(FP) 357
stat(FP) 359
syms(FP) 361
t_info(FP) 365
types(FP) 367

unistd(FP) 368
x.out(FP) 369

XENIX cross development (XNX) 373

Intro(XNX) 375
ar(XNX) 376
hdr(XNX) 378
ld(XNX) 380
nm(XNX) 384
prof(XNX) 386
ranlib(XNX) 388
size(XNX) 389
strip(XNX) 390

DOS cross development (DOS) 391

Routines(DOS) 393
abort 407
abs 407
access 407
acos 408
acosl 408
alloca 408
_arc 409
_arc_w 409
_arc_wxy 410
asctime 410
asin 411
asinl 411
assert 411
atan 412
atanl 412
atan2 413
atan2l 413
atexit 413
atof 414
atoi 414
atol 414

_atold 415
_bcalloc 415
bdos 416
_beginthread 416
_bexpand 417
_bfree 417
_bfreeseg 418
_bheapadd 418
_bheapchk 418
_bheapmin 419
_bheapseg 419
_bheapset 419
_bheapwalk 420
_bios_disk 420
_bios_equiplist 421
_bios_keybrd 421
_bios_memsize 421
_bios_printer 422
_bios_serialcom 422
_bios_timeofday 423
_bmalloc 423
_bmsize 424
_brealloc 424
bsearch 425
cabs 425
cabsl 426
calloc 426
ceil 427
ceill 427
_cexit 427
_c_exit 428
cgets 428
_chain_intr 428
chdir 429
_chdrive 429
chmod 430
chsize 430
_clear87 430
clearerr 431
_clearscreen 431
clock 431

close 432
_control87 432
cos 432
cosh 433
coshl 433
cosl 434
cprintf 434
cputs 434
creat 435
cscanf 435
ctime 436
cwait 436
dieeetomsbin 437
difftime 437
_disable 437
_displaycursor 438
div 438
dmsbintoieee 438
_dos_allocmem 439
_dos_close 439
_dos_creat 440
_dos_creatnew 440
dosexterr 441
_dos_findfirst 441
_dos_findnext 442
_dos_freemem 442
_dos_getdate 442
_dos_getdiskfree 443
_dos_getdrive 443
_dos_getfileattr 444
_dos_getftime 444
_dos_gettime 445
_dos_getvect 445
_dos_keep 445
_dos_open 446
_dos_read 446
_dos_setblock 447
_dos_setdate 447
_dos_setdrive 448
_dos_setfileattr 448
_dos_setftime 449

_dos_settime 449
_dos_setvect 450
_dos_write 450
dup 451
dup2 451
ecvt 452
_ellipse 452
_ellipse_w 453
_ellipse_wxy 453
_enable 454
_endthread 454
eof 454
execl 455
execle 455
execlp 456
execlpe 456
execv 457
execve 457
execvp 458
execvpe 458
exit 459
_exit 459
_expand 460
expl 460
fabs 461
fabsl 461
_fcalloc 461
fclose 462
fcloseall 462
fcvt 462
fdopen 463
feof 463
ferror 463
_fexpand 464
fflush 464
_ffree 464
fgetc 465
fgetchar 465
fgetpos 465
fgets 466
_fheapchk 466

_fheapmin 466
_fheapset 467
_fheapwalk 467
fieeetomsbin 468
filelength 468
fileno 468
_floodfill 469
_floodfill_w 469
floor 470
floorl 470
flushall 470
_fmalloc 471
_fmemccpy 471
_fmemchr 472
_fmemcmp 472
_fmemcpy 473
_fmemicmp 473
_fmemmove 474
_fmemset 474
fmod 475
fmodl 475
fmsbintoieee 475
_fmsize 476
fopen 476
FP_OFF 476
_fpreset 477
fprintf 477
FP_SEG 477
fputc 478
fputchar 478
fputs 478
fread 479
_frealloc 479
free 480
_freect 480
freopen 481
frexp 481
frexpl 482
fscanf 482
fseek 483
fsetpos 483

_fsopen 484
fstat 484
_fstrcat 485
_fstrchr 485
_fstrcmp 485
_fstrcpy 486
_fstrcspn 486
_fstrdup 487
_fstricmp 487
_fstrlen 487
_fstrlwr 488
_fstrncat 488
_fstrncmp 489
_fstrncpy 489
_fstrnicmp 490
_fstrnset 490
_fstrpbrk 491
_fstrrchr 491
_fstrrev 492
_fstrset 492
_fstrspn 493
_fstrstr 493
_fstrtok 494
_fstrupr 494
ftell 495
ftime 495
_fullpath 496
fwrite 496
gcvt 497
_getactivepage 497
_getarcinfo 498
_getbkcolor 498
getc 498
getch 499
getchar 499
getche 499
_getcolor 500
_getcurrentposition 500
_getcurrentposition_w 500
getcwd 501
_getdcwd 501

_getdrive 502
getenv 502
_getfillmask 502
_getfontinfo 503
_getgtextextent 503
_getgtextvetor 503
_getimage 504
_getimage_w 504
_getimage_wxy 505
_getlinestyle 505
_getphyscoord 506
getpid 506
_getpixel 506
_getpixel_w 507
gets 507
_gettextcolor 507
_gettextcursor 508
_gettextposition 508
_gettextwindow 508
_getvideoconfig 509
_getviewcoord 509
_getviewcoord_w 509
_getviewcoord_wxy 510
_getvisualpage 510
getw 510
_getwindowcoord 511
_getwritemode 511
gmtime 511
_grstatus 512
halloc 512
_harderr 513
_hardresume 513
_hardretn 513
_heapadd 514
_heapchk 514
_heapmin 514
_heapset 515
_heapwalk 515
hfree 515
hypot 516
hypotl 516

_imagesize 517
_imagesize_w 517
_imagesize_wxy 518
inp 518
inpw 519
intdos 519
intdosx 520
int86 520
int86x 521
isalnum 521
isalpha 522
isascii 522
isatty 522
iscntrl 523
isdigit 523
isgraph 523
islower 524
isprint 524
ispunct 524
isspace 525
isupper 525
isxdigit 525
itoa 526
jn 526
_jnl 527
j1 527
_j1l 527
j0 528
_j0l 528
kbhit 528
labs 529
ldexp 529
ldexpl 530
ldiv 530
lfind 531
_lineto 531
_lineto_w 532
localeconv 532
localtime 532
locking 533
log 533

logl 534
log10 534
log10l 535
longjmp 535
_lrotl 536
_lrotr 536
lsearch 537
lseek 537
ltoa 538
_makepath 538
malloc 539
matherr 539
_matherr 539
max 540
_memavl 540
memccpy 541
memchr 541
memcmp 542
memcpy 542
memicmp 543
_memmax 543
memmove 544
memset 544
min 545
mkdir 545
mktemp 545
mktime 546
modf 546
modfl 547
movedata 547
_moveto 548
_moveto_w 548
_msize 548
_ncalloc 549
_nexpand 549
_nfree 550
_nheapchk 550
_nheapmin 550
_nheapset 551
_nheapwalk 551
_nmalloc 551

_nmsize 552
_nrealloc 552
_nstrdup 552
onexit 553
open 553
_outgtext 554
_outmem 554
outp 554
outpw 555
_outtext 555
_pclose 555
perror 556
_pg_analyzechart 556
_pg_analyzechartms 557
_pg_analyzepie 558
_pg_analyzescatter 558
_pg_analyzescatterms 559
_pg_chart 559
_pg_chartms 559
_pg_chartpie 560
_pg_chartscatter 560
_pg_chartscatterms 561
_pg_defaultchart 561
_pg_getchardef 562
_pg_getpalette 562
_pg_getstyleset 562
_pg_hlabelchart 563
_pg_initchart 563
_pg_resetpalette 564
_pg_resetstyleset 564
_pg_setchardef 564
_pg_setpalette 565
_pg_setstyleset 565
_pg_vlabelchart 566
_pie 566
_pie_w 567
_pie_wxy 567
_pipe 568
_polygon 568
_polygon_w 569
_polygon_wxy 569

_popen 570
pow 570
powl 571
printf 571
putc 572
putch 572
putchar 572
putenv 573
_putimage 573
_putimage_w 574
puts 574
putw 575
qsort 575
raise 576
rand 576
read 576
realloc 577
_rectangle 577
_rectangle_w 578
_rectangle_wxy 578
_registerfonts 579
_remapallpalette 579
_remappalette 579
remove 580
rename 580
rewind 580
rmdir 581
rmtmp 581
_rotl 581
_rotr 582
scanf 582
_scrolltextwindow 583
_searchenv 583
segread 583
_selectpalette 584
_setactivepage 584
_setbkcolor 584
setbuf 585
_setcliprgn 585
_setcolor 585
_setfillmask 586

_setfont 586
_setgtextvector 586
setjmp 587
_setlinestyle 587
setlocale 588
setmode 588
_setpixel 589
_setpixel_w 589
_settextcolor 589
_settextcursor 590
_settextposition 590
_settextrows 590
_settextwindow 591
setvbuf 591
_setvideomode 592
_setvideomoderows 592
_setvieworg 593
_setviewport 593
_setvisualpage 594
_setwindow 594
_setwritemode 595
signal 595
sin 596
sinh 596
sinhl 596
sinl 597
sopen 597
spawnl 598
spawnle 598
spawnlp 599
spawnlpe 600
spawnv 600
spawnve 601
spawnvp 601
spawnvpe 602
_splitpath 602
sprintf 603
sqrt 603
sqrtl 604
srand 604
sscanf 605

stackavail 605
stat 606
_status87 606
strcat 606
strchr 607
strcmp 607
strcoll 607
strcpy 608
strcspn 608
_strdate 608
strdup 609
strerror 609
_strerror 609
strftime 610
stricmp 610
strlen 611
strlwr 611
strncat 612
strncmp 612
strncpy 613
strnicmp 613
strnset 614
strpbrk 614
strrchr 615
strrev 615
strset 615
strspn 616
strstr 616
_strtime 616
strtod 617
strtok 617
strtol 618
_strtold 618
strtoul 619
strupr 619
strxfrm 620
swab 620
system 621
tan 621
tanh 622
tanhl 622

tanl 623
tell 623
tempnam 623
time 624
tmpfile 624
tmpnam 624
toascii 625
tolower 625
_tolower 625
toupper 626
_toupper 626
tzset 626
ultoa 627
umask 627
ungetc 628
ungetch 628
unlink 628
_unregisterfonts 629
utime 629
va_arg 630
va_end 630
va_start 631
vfprintf 631
vprintf 632
vsprintf 632
wait 633
_wrapon 633
write 634
yn 634
_ynl 635
y1 635
_y1l 636
y0 636
_y0l 637

Preface

The *Programmer's Reference Manual* is a two-volume set that includes manual pages for the SCO® UNIX® Development System, including sections (CP), (FP), (DOS), (S) and (XNX).

This volume contains a complete set of the section (CP), (FP), (XNX) and (DOS) manual pages, in that order.

The manual pages for section (CP) contain comprehensive descriptions of UNIX programming commands.

The manual pages for section (FP) contain comprehensive descriptions of programming file formats.

The manual pages for section (XNX) contain comprehensive descriptions of XENIX® cross development commands.

The manual pages for section (DOS) contain comprehensive descriptions of library routines, functions, and macros used for cross development of software to DOS and OS/2® environments.

All of these manual pages are accessible online through the **man** command.

Documentation conventions

SCO documents use font changes and other typographic conventions to distinguish text elements. The following table shows these conventions:

Font conventions

Example	Entity
cc or **cc**(CP)	command. The "CP" indicates the manual page section in which the command is documented.
libc.a	filename
action	placeholder
⟨Esc⟩	keyname
open or **open**(S)	system calls, library routines, kernel functions, C keywords. The "S" indicates the manual page section in which the command is documented.
buf	structure
`b.errno`	structure member
$HOME	environment or shell variable
SIGHUP	named constant
`login`	output
"clobber"	jargon
"adm3a"	data value
date	user input

Development System documentation set

The contents of the SCO® Open Desktop® Development System documentation set are illustrated here. In addition to these books, a copy of *Release and Installation Notes* is provided with the Development System.

	Reference	Guide
UNIX® Development System	• Encyclopedia† • Programmer's Reference Manual (2 Volumes)† • Permuted Index†	• Developer's Overview† • Developer's Topics† • Programming Tools Guide† • Debugging Tools Guide† • User Interfaces Guide†
Packaging	• Software Mastering Toolkit Guide	
Networking	• Network Programmer's Guide and Reference	
Graphical Environment	• X Window System Programmer's Reference • OSF/Motif Programmer's Reference	• Integrating Applications into Open Desktop
Optional	• Macro Assembler Writer's Guide • Device Driver Writer's Guide	

† These manuals are distributed with both the UNIX Development System software and the Open Desktop Development System software.

The books included with the Development System are described here.

Developer's Overview
: introduces the Development System facilities and gives general information about developing software to run on SCO UNIX systems and the supported cross-development environments.

Developer's Topics
: a collection of technical papers about topics of interest to users of the Development System. Many of these papers include examples illustrating features that are extensions provided by SCO.

Encyclopedia
: contains articles that give background information about system internals, descriptions of facilities, and other general issues. Articles are arranged alphabetically.

Programmer's Reference Manual
: a two-volume set that includes manual pages for the entire UNIX Development System, including sections (CP), (DOS), (FP), (S), and (XNX). See the "Manual pages" article in the *Encyclopedia* for a description of these manual page sections.

Permuted Index

a permuted index of all manual pages that make up the Open Desktop Development System.

Programming Tools Guide

provides generally useful information about programming tools and their use in developing software. It also provides implementation-specific details about the ANSI-conforming C compiler provided with the Development System.

Debugging Tools Guide

provides generally useful information about the debugging tools, and their use in tracking down and eliminating problems in C and assembly language programs. In addition, it includes information on using **dbXtra**™ to debug C and C++ programs in an OSF/Motif™-based windowing environment.

User Interfaces Guide

introduces the facilities available for developing user interfaces, and gives detailed instructions for using **curses**(S) and the Extended Terminal Interface (ETI), and for writing user interfaces that can be run on ASCII terminals. This book includes numerous examples of **curses** programs.

Software Mastering Toolkit Guide

provides information and examples for using the Software Mastering Toolkit® to develop a **custom**-installable product. Also included are the (SMT) manual pages.

Network Programmer's Guide and Reference

provides procedures and reference information for developing applications that use the SCO networking services: TCP/IP, NFS™, IPX/SPX, TLI/XTI, int5c, and the PC-Interface™ extended library. As well, the (SLIB), (SSC), (NC), (NS), and (PCI) man pages are provided in this volume.

X Window System Programmer's Reference

provides X Window System™ programming commands and functions: X Library (XS), X Toolkit (Xt), X Miscellaneous Utilities (Xmu), and X Extensions (Xext) reference manual pages for programmers.

OSF/Motif Programmer's Reference

consists of the OSF/Motif toolkit, window manager, and user interface language commands and functions: (Xm) reference manual pages.

Integrating Applications into Open Desktop

provides information about application programming interfaces that help in integrating applications into the Open Desktop

graphical environment: putting an icon on the desktop, adapting to display resolution at runtime, supporting the desktop color selector, and communicating with the session manager.

In addition, the following books are available on request:

Macro Assembler Writer's Guide
: provides information on the assembly language for the Intel® 286, 386, and 486 processors. Developers who are writing device drivers or applications having tight performance requirements may want to write portions in assembly language. This book is sold separately.

Device Driver Writer's Guide
: provides guide and reference information about writing device drivers for SCO UNIX systems. This book is sold separately.

Using the Documentation

We recommend the following approach to using this documentation set. In general, we suggest the following:

1. All users should read the *Developer's Overview* thoroughly, soon after installing the software. Experienced users may skim some sections, but this book gives a comprehensive overview of the Development System's capabilities.

2. Use the *Encyclopedia* to look up topics about which you need more information. The *Encyclopedia* gives both summary and background information on many topics, and includes cross-references to sources of more in-depth information.

3. The *Programmer's Reference Manual* contains complete reference information about all capabilities provided in the UNIX Development System. New users are warned, however, that the manual pages provide reference material that may be difficult to utilize if they do not understand basic concepts.

   The *Programmer's Reference Manual* contains separate sections for Development System commands (CP), system services (S), Development System file formats (FP), and other topics. The pages in each section of the *Programmer's Reference Manual* are arranged alphabetically, although some related facilities are grouped together on a page. If you cannot locate the correct manual page, use the permuted index to find the one that contains the information you want. The "Manual pages" article in the *Encyclopedia* explains how to use the permuted index.

4. The *Programming Tools Guide* and *Debugging Tools Guide* are mainly for users who have not used one of these tools before, or who want to

reacquaint themselves with some aspect of that tool's operation. These books are good places to look when the *Encyclopedia* and *Programmer's Reference Manual* are giving you lots of detail, but not enough guidance. Nonetheless, even experienced programmers will discover helpful hints and practical tips on the tools discussed in these books.

Commercial books and articles

A number of fine books and articles are published commercially that discuss how to develop software on the UNIX Operating System. We have not attempted to replicate guide information on all of these topics; manual pages are provided, and many articles in the *Encyclopedia* list additional sources of information on their subject matters. In addition, the *Encyclopedia* includes a large "Bibliography" that will interest users of the Development System.

In particular, we recommend that all developers have the following standard textbooks on their shelves:

Brian Kernighan and Dennis Ritchie, *The C Programming Language*, 2nd Edition.

One of the following general books about programming on UNIX operating systems:

Donald Lewine, *POSIX Programmer's Guide.*

Marc Rochkind, *Advanced UNIX Programming.*

W. Richard Stevens, *Advanced Programming in the UNIX Environment.*

W. Richard Stevens, *UNIX Network Programming.* Useful for programmers who are writing distributed applications.

Maurice Bach, *Design of the UNIX Operating System.* Useful for programmers who want to thoroughly understand UNIX system internals.

See the "Bibliography" in the *Encyclopedia* for full citations if you are not familiar with the books listed above.

Reference manual sections

Reference material is distributed as individual reference sections in the various volumes of the Operating and Development Systems. The following table lists the section name, description, and location of each reference section.

Section	Description	Volume
ADM	administrative commands	*System Administrator's Reference*
C	commands	*User's Reference*
CP	programming commands	*Programmer's Reference Manual*
DOS	DOS and OS/2® library routines	*Programmer's Reference Manual*
F	file formats	*System Administrator's Reference*
FP	programming file formats	*Programmer's Reference Manual*
HW	hardware devices	*System Administrator's Reference*
K	kernel functions used in device drivers	*Device Driver Writer's Guide*
M	miscellaneous	*User's Reference*
NC	RPC protocol compiler	*Network Programmer's Guide and Reference*
NS	network system calls	*Network Programmer's Guide and Reference*
PCI	PC-Interface extended library	*Network Programmer's Guide and Reference*
S	system calls and library routines	*Programmer's Reference Manual*
SLIB	socket library functions	*Network Programmer's Guide and Reference*
SMT	Software Mastering Toolkit utilities	*Software Mastering Toolkit Guide*
SSC	socket system calls	*Network Programmer's Guide and Reference*
X	X clients	online only
XNX	XENIX® cross development	*Programmer's Reference Manual*
XS	X library routines	*X Window System Programmer's Reference*
Xext	X Extensions library	*X Window System Programmer's Reference*
Xm	OSF/Motif commands and functions	*OSF/Motif Programmer's Reference*
Xmu	Xmu library	*X Window System Programmer's Reference*
Xt	X Toolkit Intrinsics library	*X Window System Programmer's Reference*

The alphabetized list that follows is a complete listing of all UNIX commands, system calls, library routines, and file formats: (ADM), (C), (CP), (DOS), (F), (FP), (HW), (M), (S), (XNX).

Alphabetized list

Commands, system calls, library routines, and file formats

[..... test(C)
300 ..... 300(C)
300s ..... 300 (C)
4014 ..... 4014(C)
450 ..... 450 (C)
80387 ..... 80387 (HW)
86rel ..... 86rel(FP)
a64l ..... a64l (S)
abort ..... abort (S)
abs ..... abs (S)
accept ..... accept (ADM)
acceptable_password ..... accept_pw (S)
access ..... access (S)
acct ..... acct (ADM)
acct ..... acct (FP)
acct ..... acct (S)
acctcms ..... acctcms (ADM)
acctcom ..... acctcom (ADM)
acctcon ..... acctcon (ADM)
acctcon1 ..... acctcon (ADM)
acctcon2 ..... acctcon (ADM)
acctdisk ..... acct (ADM)
acctdusg ..... acct (ADM)
acctmerg ..... acctmerg (ADM)
accton ..... acct (ADM)
accton ..... accton (ADM)
acctprc ..... acctprc (ADM)
acctprc1 ..... acctprc (ADM)
acctprc2 ..... acctprc (ADM)
acctsh ..... acctsh (ADM)
acctwtmp ..... acct (ADM)
acos ..... trig (S)
adb ..... adb (CP)
addch ..... curses (S)
addch ..... tam (S)
addch ..... terminfo (S)
addkey ..... curses (S)
addkey ..... terminfo (S)
addstr ..... curses(S)
addstr ..... tam (S)
addstr ..... terminfo (S)
addxusers ..... addxusers (ADM)
adf_gttok ..... tam (S)
adf_gtwrd ..... tam (S)
adf_gtxcd ..... tam (S)
admin ..... admin (CP)
advance ..... regexp (S)
agetcommand ..... authcap (S)
agetdefault ..... authcap (S)
agetfile ..... authcap (S)
agetflag ..... authcap (S)
agetstr ..... authcap (S)
agettty ..... authcap (S)
agetuser ..... authcap (S)
aio ..... aio (F)
aio ..... aio (M)
aioinfo ..... aioinfo (ADM)
aiolkinit ..... aiolkinit (ADM)
aiomemlock ..... aiomemlock (F)
alarm ..... alarm (S)
ale ..... ale (ADM)
allocldptr ..... ldptr (S)
altzone ..... ctime (S)
a.out ..... a.out (FP)
ap ..... ap (ADM)
ar ..... ar (CP)
ar ..... ar (FP)
ar ..... ar (XNX)
arc ..... plot (S)
archive ..... archive (F)
as ..... as (CP)
asa ..... undocumented (M)
ascii ..... ascii (M)
asctime ..... ctime (S)
asetdefaults ..... authcap (S)
asin ..... trig (S)
asktime ..... asktime (ADM)
asktimer ..... asktime (ADM)
asroot ..... asroot (ADM)
assert ..... assert (S)
assign ..... assign (C)
at ..... at (C)
atan ..... trig (S)
atan2 ..... trig (S)
atcronsh ..... atcronsh (ADM)
atexit ..... atexit (S)
atof ..... atof (S)
atoi ..... atof (S)
atol ..... atof (S)
attroff ..... curses (S)

attroff ..... tam (S)
attroff ..... terminfo (S)
attron ..... curses (S)
attron ..... tam (S)
attron ..... terminfo (S)
attrset ..... curses (S)
attrset ..... terminfo (S)
audit ..... audit(HW)
audit_adjust_mask ..... authaudit (S)
audit_auth_entry ..... authaudit (S)
audit_close ..... audit (S)
auditcmd ..... auditcmd (ADM)
auditd ..... auditd (ADM)
audit_lax_file ..... authaudit (S)
audit_lock ..... authaudit (S)
audit_login ..... authaudit (S)
audit_no_resource ..... authaudit (S)
audit_open ..... audit (S)
audit_passwd ..... authaudit (S)
audit_read ..... audit (S)
audit_security_failure ..... authaudit (S)
auditsh ..... auditsh (ADM)
audit_subsystem ..... authaudit (S)
authaudit ..... authaudit (S)
authcap ..... authcap (F)
authcap ..... authcap (S)
authck ..... authck (ADM)
authckrc ..... tcbck (ADM)
authorize ..... authorize (F)
authorized_user ..... subsystems (S)
auths ..... auths (C)
authsh ..... authsh (ADM)
autoboot ..... autoboot (ADM)
awk ..... awk (C)
backup ..... backup (ADM)
backupsh ..... backupsh (ADM)
badtrk ..... badtrk (ADM)
banner ..... banner(C)
basename ..... basename (C)
batch ..... at (C)
baudrate ..... curses (S)
baudrate ..... tam (S)
baudrate ..... terminfo (S)
bc ..... bc (C)
BC ..... termcap (S)
bcheckrc ..... bcheckrc (ADM)
bdiff ..... bdiff(C)
beep ..... curses (S)
beep ..... tam (S)
beep ..... terminfo (S)
bessel ..... bessel (S)
bfs ..... bfs (C)
bigcrypt ..... getpasswd (S)
bigcryptmax ..... getpasswd (S)
boot ..... boot (HW)
Bottom ..... libwindows (S)
bottom_panel ..... panel (S)
box ..... curses (S)
box ..... plot (S)
box ..... terminfo (S)
brand ..... undocumented (M)
brk ..... brk (S)
brkctl ..... brkctl (S)
bsearch ..... bsearch (S)
btld ..... btld (F)
btldinstall ..... btldinstall (ADM)
cal ..... cal (C)
calendar ..... calendar (C)
calloc ..... malloc (S)
cancel ..... cancel (C)
can_change_color ..... curses (S)
can_change_color ..... terminfo (S)
captoinfo ..... captoinfo (ADM)
cat ..... cat (C)
catclose ..... catopen (S)
catgets ..... catgets (S)
catopen ..... catopen (S)
cb ..... cb (CP)
cbackup ..... cbackup (ADM)
cbreak ..... curses (S)
cbreak ..... tam (S)
cbreak ..... terminfo (S)
cc ..... cc (CP)
cd ..... cd (C)
cdc ..... cdc (CP)
cdrom ..... cdrom (HW)
ceil ..... floor (S)
cfgetispeed ..... cfspeed (S)
cfgetospeed ..... cfspeed (S)
cflow ..... cflow (CP)
cfree ..... malloc (S)
cfsetispeed ..... cfspeed (S)
cfsetospeed ..... cfspeed (S)
cfspeed ..... cfspeed (S)
chargefee ..... acctsh (ADM)
chdir ..... chdir (S)
checkaddr ..... checkaddr (ADM)
check_auth_parameters ..... identity (S)
checkeq ..... undocumented (M)
checklist ..... checklist (F)
checkmail ..... checkmail (C)
checkque ..... checkque (ADM)

checkup ..... checkup (ADM)
chg_audit ..... chg_audit (ADM)
chgrp ..... chgrp (C)
chkshlib ..... chkshlib (CP)
chmod ..... chmod (C)
chmod ..... chmod (S)
chown ..... chown (C)
chown ..... chown (S)
chroot ..... chroot (ADM)
chroot ..... chroot (S)
chrtbl ..... chrtbl (M)
chsize ..... chsize (S)
chtype ..... unretire (ADM)
circf ..... regexp (S)
circle ..... plot (S)
ckbupscd ..... undocumented (M)
ckpacct ..... acctsh (ADM)
cksum ..... cksum (C)
cleanque ..... cleanque (ADM)
cleantmp ..... cleantmp (ADM)
cleanup ..... undocumented (M)
clear ..... clear (C)
clear ..... curses (S)
clear ..... tam (S)
clear ..... terminfo (S)
clearerr ..... ferror (S)
clearok ..... curses (S)
clearok ..... tam (S)
clearok ..... terminfo (S)
clock ..... clock (F)
clock ..... clock (S)
clone ..... clone (M)
close ..... close (S)
closedir ..... directory (S)
closepl ..... plot (S)
clri ..... clri (ADM)
clrtobot ..... curses (S)
clrtobot ..... tam (S)
clrtobot ..... terminfo (S)
clrtoeol ..... curses (S)
clrtoeol ..... tam (S)
clrtoeol ..... terminfo (S)
cmos ..... cmos (HW)
cmp ..... cmp (C)
cnvtmbox ..... cnvtmbox (ADM)
codeview ..... codeview (CP)
col ..... col (C)
color_content ..... curses (S)
color_content ..... terminfo (S)
coltbl ..... coltbl (M)
comb ..... comb (CP)
comm ..... comm (C)
compile ..... regexp (S)
compress ..... compress (C)
compver ..... compver (F)
configure ..... configure (ADM)
console ..... console (M)
consoleprint ..... consoleprint (ADM)
cont ..... plot (S)
conv ..... conv (CP)
convert ..... convert (CP)
convkey ..... mapkey (M)
copy ..... copy (C)
copydvagent ..... getdvagent (S)
copyright ..... copyright (F)
copywin ..... curses (S)
copywin ..... terminfo (S)
core ..... core (FP)
corex ..... corex (C)
cos ..... trig (S)
cosh ..... sinh (S)
cp ..... cp (C)
cpio ..... cpio (C)
cpio ..... cpio (F)
cpp ..... cpp (CP)
cprs ..... cprs (CP)
cps ..... fixmog (ADM)
cpset ..... cpset (C)
crash ..... crash (ADM)
creat ..... creat (S)
create_file_securely ... create_file_securely (S)
creatsem ..... creatsem (S)
crmode ..... curses (S)
crmode ..... tam (S)
crmode ..... terminfo (S)
cron ..... cron (C)
crontab ..... crontab (C)
crypt ..... crypt (C)
crypt ..... crypt (S)
crypt_close ..... crypt (S)
cryptopen ..... crypt (S)
cscope ..... cscope (CP)
csh ..... csh (C)
csplit ..... csplit (C)
ct ..... ct (C)
ctags ..... ctags (C)
ctermid ..... ctermid (S)
ctime ..... ctime (S)
ctrace ..... ctrace (CP)
ctype ..... ctype (S)
cu ..... cu (C)
curoff ..... curses (S)

curoff ..... terminfo(S)
curon ..... curses(S)
curon ..... terminfo(S)
Current ..... libwindows(S)
current_field ..... form(S)
current_item ..... menu(S)
curses ..... curses(S)
curs_set ..... curses(S)
curs_set ..... terminfo(S)
curtbl ..... montbl(M)
cuserid ..... cuserid(S)
custom ..... custom(ADM)
cut ..... cut(C)
cv ..... codeview(CP)
cxref ..... cxref(CP)
dat ..... dat(HW)
date ..... date(C)
daylight ..... ctime(S)
dblock ..... dblock(S)
dbm ..... dbm(S)
dbmbuild ..... dbmbuild(ADM)
dbmedit ..... dbmedit(ADM)
dbminit ..... dbm(S)
dbXtra ..... dbXtra(CP)
dc ..... dc(C)
dcopy ..... dcopy(ADM)
dd ..... dd(C)
deassign ..... assign(C)
debrand ..... undocumented(M)
default ..... default(F)
defaults ..... defaults(F)
defopen ..... defopen(S)
def_prog_mode ..... curses(S)
def_prog_mode ..... terminfo(S)
defread ..... defopen(S)
def_shell_mode ..... curses(S)
def_shell_mode ..... terminfo(S)
delay_output ..... curses(S)
delay_output ..... terminfo(S)
delch ..... curses(S)
delch ..... tam(S)
delch ..... terminfo(S)
del_curterm ..... curses(S)
del_curterm ..... terminfo(S)
delete ..... dbm(S)
Delete ..... libwindows(S)
deleteln ..... curses(S)
deleteln ..... tam(S)
deleteln ..... terminfo(S)
deliver ..... deliver(ADM)
del_panel ..... panel(S)
delta ..... delta(CP)
delwin ..... curses(S)
delwin ..... terminfo(S)
depend ..... depend(F)
des_crypt ..... crypt(S)
des_encrypt ..... crypt(S)
des_setkey ..... crypt(S)
devassign ..... devassign(F)
devices ..... devices(F)
devnm ..... devnm(C)
df ..... df(C)
dfsck ..... fsck(ADM)
dfspace ..... dfspace(C)
dial ..... dial(ADM)
dial ..... dial(S)
dialcodes ..... dialcodes(F)
dialers ..... dialers(F)
diff ..... diff(C)
diff3 ..... diff3(C)
difftime ..... difftime(S)
dir ..... dir(FP)
dircmp ..... dircmp(C)
directory ..... directory(S)
dirent ..... dirent(FP)
dirname ..... dirname(C)
dis ..... dis(CP)
disable ..... disable(C)
diskcmp ..... diskcp(C)
diskcp ..... diskcp(C)
diskusg ..... diskusg(ADM)
displaypkg ..... displaypkg(ADM)
div ..... div(S)
divvy ..... divvy(ADM)
dkinit ..... dparam(ADM)
dlvr_audit ..... dlvr_audit(ADM)
dmesg ..... dmesg(ADM)
dmp_win ..... curses(S)
dodisk ..... acctsh(ADM)
doscat ..... doscmd(C)
doscmd ..... doscmd(C)
doscp ..... doscmd(C)
dosdir ..... doscmd(C)
dosformat ..... doscmd(C)
dosld ..... dosld(CP)
dosls ..... doscmd(C)
dosmkdir ..... doscmd(C)
dosrm ..... doscmd(C)
dosrmdir ..... doscmd(C)
doupdate ..... curses(S)
doupdate ..... terminfo(S)
dparam ..... dparam(ADM)

draino ..... curses (S)
draino ..... terminfo (S)
drand48 ..... drand48 (S)
dsconfig ..... undocumented (M)
dtox ..... dtox (C)
dtype ..... dtype (C)
du ..... du (C)
dump ..... dump (CP)
dumpmsg ..... dumpmsg (CP)
dumpwin ..... terminfo (S)
dup ..... dup (S)
dup2 ..... dup2 (S)
dup_field ..... field (S)
dupwin ..... curses (S)
dupwin ..... terminfo (S)
eaccess ..... access (S)
ecc ..... ecc (ADM)
eccd ..... ecc (ADM)
echo ..... curses (S)
echo ..... echo (C)
echo ..... tam (S)
echo ..... terminfo (S)
echochar ..... curses (S)
echochar ..... terminfo (S)
ecvt ..... ecvt (S)
ed ..... ed (C)
edata ..... end (S)
edit ..... ex (C)
egrep ..... grep (C)
eisa ..... eisa (ADM)
emactovi ..... undocumented (M)
enable ..... enable (C)
encrypt ..... crypt (S)
end ..... end (S)
enddvagent ..... getdvagent (S)
endgrent ..... getgrent (S)
endprdfent ..... getprdfent (S)
endprfient ..... getprfient (S)
endprpwent ..... getprpwent (S)
endprtcent ..... getprtcent (S)
endpwent ..... getpwent (S)
endspent ..... getspent (S)
endutent ..... getut (S)
endwin ..... curses (S)
endwin ..... tam (S)
endwin ..... terminfo (S)
enter_quiet_zone ..... dblock (S)
env ..... env (C)
environ ..... environ (M)
erand48 ..... drand48 (S)
erase ..... curses (S)
erase ..... plot (S)
erase ..... tam (S)
erase ..... terminfo (S)
erasechar ..... curses (S)
erasechar ..... terminfo (S)
erf ..... erf (S)
erfc ..... erf (S)
errno ..... perror (S)
error ..... error (M)
ERROR ..... regexp (S)
etext ..... end (S)
ev_block ..... ev_block (S)
ev_close ..... ev_close (S)
ev_count ..... ev_count (S)
ev_flush ..... ev_flush (S)
ev_getdev ..... ev_getdev (S)
ev_getemask ..... ev_getemask (S)
ev_gindev ..... ev_gindev (S)
ev_init ..... ev_init (S)
ev_initf ..... ev_init (S)
ev_open ..... ev_open (S)
ev_pop ..... ev_pop (S)
ev_read ..... ev_read (S)
ev_resume ..... ev_resume (S)
ev_setemask ..... ev_setemask (S)
ev_suspend ..... ev_suspend (S)
ex ..... ex (C)
exec ..... exec (S)
execl ..... exec (S)
execle ..... exec (S)
execlp ..... exec (S)
execseg ..... execseg (S)
execv ..... exec (S)
execve ..... exec (S)
execvp ..... exec (S)
exhelp ..... tam (S)
exit ..... exit (S)
Exit ..... libwindows (S)
exit_quiet_zone ..... dblock (S)
exp ..... exp (S)
expand ..... expand (C)
expr ..... expr (C)
fabs ..... floor (S)
factor ..... factor (C)
false ..... false (C)
fclose ..... fclose (S)
fcntl ..... fcntl (M)
fcntl ..... fcntl (S)
fcvt ..... ecvt (S)
fd ..... fd (HW)
FD_CLR ..... select (S)

fdisk ..... fdisk (ADM)
FD_ISSET ..... select (S)
fdopen ..... fopen (S)
FD_SET ..... select (S)
fdswap ..... fdswap (ADM)
FD_ZERO ..... select (S)
feof ..... ferror (S)
ferror ..... ferror (S)
fetch ..... dbm (S)
ff ..... ff(ADM)
fflush ..... fclose (S)
fgetc ..... getc (S)
fgetgrent ..... getgrent (S)
fgetpasswd ..... getpasswd (S)
fgetpos ..... fgetpos (S)
fgetpwent ..... getpwent (S)
fgets ..... gets (S)
fgetspent ..... getspent (S)
fgrep ..... grep (C)
field ..... field (S)
field_arg ..... field (S)
field_back ..... field (S)
field_buffer ..... field (S)
field_count ..... form (S)
field_fore ..... field (S)
field_index ..... form (S)
field_info ..... field (S)
field_init ..... form (S)
field_just ..... field (S)
field_opts ..... field (S)
field_opts_off ..... field (S)
field_opts_on ..... field (S)
field_pad ..... field (S)
fields ..... fields (S)
field_status ..... field (S)
field_term ..... form (S)
field_type ..... field (S)
fieldtype ..... fieldtype (S)
field_userptr ..... field (S)
file ..... file (C)
filehdr ..... filehdr (FP)
fileno ..... ferror (S)
files ..... files (F)
filesys ..... filesys (F)
filesystem ..... filesystem (FP)
filter ..... curses (S)
filter ..... terminfo (S)
find ..... find (C)
findstr ..... findstr (CP)
finger ..... finger (C)
firstkey ..... dbm (S)
fixhdr ..... fixhdr (C)
fixmog ..... fixmog (ADM)
fixperm ..... fixperm (ADM)
fixshlib ..... undocumented (M)
fixterm ..... curses (S)
fixterm ..... tam (S)
flash ..... curses (S)
flash ..... tam (S)
flash ..... terminfo (S)
floor ..... floor (S)
flushinp ..... curses (S)
flushinp ..... tam (S)
flushinp ..... terminfo (S)
fmod ..... floor(S)
fold ..... fold (C)
fopen ..... fopen (S)
fork ..... fork (S)
form ..... form (S)
form ..... tam (S)
format ..... format (C)
form_driver ..... form (S)
form_fields ..... form (S)
form_init ..... form (S)
form_opts ..... form (S)
form_opts_off ..... form (S)
form_opts_on ..... form (S)
form_page ..... form (S)
form_sub ..... form (S)
form_term ..... form (S)
form_userptr ..... form (S)
form_win ..... form (S)
fpathconf ..... pathconf (S)
fpgetmask ..... fpgetround (S)
fpgetround ..... fpgetround (S)
fpgetsticky ..... fpgetround (S)
fprintf ..... printf (S)
fpsetmask ..... fpgetround (S)
fpsetround ..... fpgetround (S)
fpsetsticky ..... fpgetround (S)
fputc ..... putc (S)
fputs ..... puts (S)
fread ..... fread (S)
frec ..... undocumented (M)
free ..... malloc (S)
free_field ..... field (S)
free_fieldtype ..... fieldtype (S)
free_form ..... form (S)
free_item ..... item (S)
freeldptr ..... ldptr (S)
free_menu ..... menu (S)
freopen ..... fopen (S)

frexp ..... frexp (S)
fsanck ..... undocumented (M)
fsave ..... fsave (ADM)
fsba ..... undocumented (M)
fscanf ..... scanf (S)
fsck ..... fsck (ADM)
fsdb ..... fsdb (ADM)
fseek ..... fseek (S)
fsetpos ..... fsetpos (S)
fsname ..... fsname (ADM)
fspec ..... fspec (F)
fsphoto ..... fsphoto (ADM)
fsstat ..... fsstat (ADM)
_fst ..... undocumented (M)
fstat ..... stat (S)
fstatfs ..... statfs (S)
fstyp ..... fstyp (ADM)
fsync ..... fsync (S)
ftell ..... fseek (S)
ftime ..... time (S)
ftok ..... ftok (S)
ftw ..... ftw (S)
fuser ..... fuser (ADM)
fwrite ..... fread (S)
fwtmp ..... fwtmp (ADM)
fxlist ..... xlist (S)
gamma ..... gamma (S)
garbagedlines ..... curses(S)
garbagedlines ..... terminfo (S)
gcvt ..... ecvt (S)
gencat ..... gencat (CP)
gencc ..... gencc (CP)
get ..... get (CP)
getbegyx ..... curses (S)
getbegyx ..... terminfo (S)
getc ..... getc (S)
GETC ..... regexp (S)
getch ..... curses (S)
getch ..... tam (S)
getch ..... terminfo (S)
getchar ..... getc (S)
getclk ..... getclk (M)
getconf ..... getconf (C)
getcwd ..... getcwd (S)
getdents ..... getdents (S)
getdim ..... curses (S)
getdim ..... terminfo (S)
getdvagent ..... getdvagent (S)
getdvagnam ..... getdvagent (S)
getegid ..... getuid (S)
getenv ..... getenv (S)
geteuid ..... getuid (S)
getgid ..... getuid (S)
getgrent ..... getgrent (S)
getgrgid ..... getgrent (S)
getgrnam ..... getgrent (S)
getgroups ..... getgroups (S)
gethz ..... gethz (S)
getitimer ..... getitimer (S)
getlogin ..... getlogin (S)
getluid ..... getluid (S)
getmaxyx ..... curses (S)
getmaxyx ..... terminfo (S)
getmsg ..... getmsg (S)
getopt ..... getopt (C)
getopt ..... getopt (S)
getoptcvt ..... getopts (C)
getopts ..... getopts (C)
getorg ..... curses (S)
getorg ..... terminfo (S)
getpass ..... getpass (S)
getpasswd ..... getpasswd (S)
getpgrp ..... getpid (S)
getpid ..... getpid (S)
getppid ..... getpid (S)
getprdfent ..... getprdfent (S)
getprdfnam ..... getprdfent (S)
getprfient ..... getprfient (S)
getprfinam ..... getprfient (S)
getpriv ..... getpriv (S)
getprpwent ..... getprpwent (S)
getprpwnam ..... getprpwent (S)
getprpwuid ..... getprpwent (S)
getprtcent ..... getprtcent (S)
getprtcnam ..... getprtcent (S)
getpw ..... getpw (S)
getpwent ..... getpwent (S)
getpwnam ..... getpwent (S)
getpwuid ..... getpwent (S)
gets ..... gets (C)
gets ..... gets (S)
get_seed ..... seed (S)
getserno ..... getserno (C)
getspent ..... getspent (S)
getspnam ..... getspent (S)
getstr ..... curses (S)
getstr ..... terminfo (S)
getsyx ..... curses (S)
getsyx ..... terminfo (S)
gettmode ..... curses (S)
getty ..... getty (M)
gettydefs ..... gettydefs (F)

getuid ..... getuid (S)
getut ..... getut (S)
getutent ..... getut (S)
getutid ..... getut (S)
getutline ..... getut (S)
getw ..... getc (S)
getyx ..... curses (S)
getyx ..... tam (S)
getyx ..... terminfo (S)
gmtime ..... ctime (S)
goodpw ..... goodpw (ADM)
graph ..... graph (ADM)
greek ..... greek (C)
grep ..... grep (C)
gr_idtoname ..... pw_nametoid (S)
gr_nametoid ..... pw_nametoid (S)
group ..... group (F)
grpck ..... grpck (ADM)
gsignal ..... ssignal (S)
halfdelay ..... curses (S)
halfdelay ..... terminfo (S)
haltsys ..... haltsys (ADM)
has_colors ..... curses (S)
has_colors ..... terminfo (S)
hashcheck ..... spell (C)
hashmake ..... spell (C)
has_ic ..... curses (S)
has_ic ..... terminfo (S)
has_il ..... curses (S)
has_il ..... terminfo (S)
hcreate ..... hsearch (S)
hd ..... hd (C)
hd ..... hd (HW)
hdestroy ..... hsearch (S)
hdr ..... hdr (XNX)
head ..... head (C)
hello ..... hello (C)
help ..... help (CP)
hide_panel ..... panel (S)
hp ..... hp (C)
hs ..... hs (F)
hsearch ..... hsearch (S)
hwconfig ..... hwconfig (C)
hypot ..... hypot (S)
i286 ..... machid (C)
i286emul ..... i286emul (C)
i286emul ..... i286emul (CP)
i386 ..... machid (C)
i486 ..... machid (C)
iAPX286 ..... machid (C)
ibmlpopt ..... undocumented (M)
iconv ..... iconv (CP)
id ..... id (C)
idaddld ..... idaddld (ADM)
idas ..... idas (M)
idbuild ..... idbuild (ADM)
idcheck ..... idcheck (ADM)
idconfig ..... idbuild (ADM)
identity ..... identity (S)
idinstall ..... idinstall (ADM)
idld ..... idld (M)
idleout ..... idleout (ADM)
idlok ..... curses (S)
idlok ..... terminfo (S)
idmkenv ..... idbuild (ADM)
idmkinit ..... idmkinit (ADM)
idmknod ..... idmknod (ADM)
idmkunix ..... idbuild (ADM)
idscsi ..... idbuild (ADM)
idspace ..... idspace (ADM)
idtune ..... idtune (ADM)
idvidi ..... idbuild (ADM)
inch ..... curses (S)
inch ..... terminfo (S)
infocmp ..... infocmp (ADM)
inipcrm ..... undocumented (M)
init ..... init (M)
INIT ..... regexp (S)
init.base ..... inittab (F)
init_color ..... curses (S)
init_color ..... terminfo (S)
initcond ..... initcond (ADM)
init_pair ..... curses (S)
init_pair ..... terminfo (S)
initscr ..... curses (S)
initscr ..... tam (S)
initscr ..... terminfo (S)
initscript ..... initscript (ADM)
inittab ..... inittab (F)
inode ..... inode (FP)
insch ..... curses (S)
insch ..... tam (S)
insch ..... terminfo (S)
insertln ..... curses (S)
insertln ..... tam (S)
insertln ..... terminfo (S)
insertmsg ..... insertmsg (CP)
install ..... install (ADM)
installf ..... installf (ADM)
installpkg ..... installpkg (ADM)
integrity ..... integrity (ADM)
intrflush ..... curses (S)

intrflush ..... terminfo (S)
Intro ..... Intro (ADM)
Intro ..... Intro (C)
Intro ..... Intro (CP)
Intro ..... Intro (F)
Intro ..... Intro (FP)
Intro ..... Intro (HW)
Intro ..... Intro (M)
Intro ..... Intro (S)
Intro ..... Intro (XNX)
ioctl ..... ioctl (S)
ipcrm ..... ipcrm (ADM)
ipcs ..... ipcs (ADM)
isaddindex ..... isaddindex (S)
isalnum ..... ctype (S)
isalpha ..... ctype (S)
isascii ..... ctype (S)
isatty ..... ttyname (S)
isbuild ..... isbuild (S)
isclose ..... isclose (S)
iscntrl ..... ctype (S)
isconv ..... isconv (S)
isdelcurr ..... isdelcurr (S)
isdelete ..... isdelete (S)
isdelindex ..... isdelindex (S)
isdelrec ..... isdelrec (S)
isdigit ..... ctype (S)
isendwin ..... curses (S)
isendwin ..... terminfo (S)
iserase ..... iserase (S)
isgraph ..... ctype (S)
isindexinfo ..... isindexinfo (S)
islock ..... islock (S)
islower ..... ctype (S)
ismpx ..... ismpx (C)
isnan ..... isnan (S)
isnand ..... isnan (S)
isnanf ..... isnan (S)
isopen ..... isopen (S)
isprint ..... ctype (S)
ispunct ..... ctype (S)
isread ..... isread (S)
isrelease ..... isrelease (S)
isrename ..... isrename (S)
isrewcurr ..... isrewcurr (S)
isrewrec ..... isrewrec (S)
isrewrite ..... isrewrite (S)
issetunique ..... issetunique (S)
isspace ..... ctype (S)
isstart ..... isstart (S)
is_starting_egid ..... identity (S)
is_starting_euid ..... identity (S)
is_starting_luid ..... identity (S)
is_starting_rgid ..... identity (S)
is_starting_ruid ..... identity (S)
issue ..... issue (F)
isuniqueid ..... isuniqueid (S)
isunlock ..... isunlock (S)
isupper ..... ctype (S)
isverify ..... isverify (M)
iswind ..... tam (S)
iswrcurr ..... iswrcurr (S)
iswrite ..... iswrite (S)
isxdigit ..... ctype (S)
item ..... item (S)
item_count ..... item (S)
item_description ..... item (S)
item_index ..... menu (S)
item_init ..... menu (S)
item_name ..... item (S)
item_opts ..... item (S)
item_opts_off ..... item (S)
item_opts_on ..... item (S)
item_term ..... menu (S)
item_userptr ..... item (S)
item_value ..... item (S)
item_visible ..... item (S)
itimer ..... getitimer (S)
j0 ..... bessel (S)
j1 ..... bessel (S)
jagent ..... jagent (M)
jn ..... bessel (S)
join ..... join (C)
jrand48 ..... drand48 (S)
jterm ..... jterm (C)
jwin ..... jwin (C)
kbmode ..... kbmode (ADM)
kcodemap ..... tam (S)
keyboard ..... keyboard (HW)
keyname ..... curses (S)
keyname ..... terminfo (S)
keypad ..... curses (S)
keypad ..... tam (S)
keypad ..... terminfo (S)
kill ..... kill (C)
kill ..... kill (S)
killall ..... killall (ADM)
killchar ..... curses (S)
killchar ..... terminfo (S)
kmem ..... mem (FP)
ksh ..... ksh (C)
l ..... ls (C)

l3tol ..... l3tol (S)
l64a ..... a64l (S)
label ..... plot (S)
labelit ..... labelit (ADM)
labs ..... labs (S)
langinfo ..... langinfo (FP)
last ..... last (C)
lastlogin ..... acctsh (ADM)
layers ..... layers (C)
layers ..... layers (M)
lc ..... ls (C)
lckpwdf ..... getspent (S)
lcong48 ..... drand48 (S)
lconv ..... lconv (FP)
ld ..... ld (CP)
ld ..... ld (XNX)
ldaclose ..... ldclose (S)
ldahread ..... ldahread (S)
ldaopen ..... ldopen (S)
ldclose ..... ldclose (S)
lddbl ..... isconv (S)
ldexp ..... frexp (S)
ldfcn ..... ldfcn (FP)
ldfhread ..... ldfhread (S)
ldfloat ..... isconv (S)
ldgetname ..... ldgetname (S)
ldint ..... isconv (S)
ldiv ..... ldiv (S)
ldlinit ..... ldlread (S)
ldlitem ..... ldlread (S)
ldlong ..... isconv (S)
ldlread ..... ldlread (S)
ldlseek ..... ldlseek (S)
ldnlseek ..... ldlseek (S)
ldnrseek ..... ldrseek (S)
ldnshread ..... ldshread (S)
ldnsseek ..... ldsseek (S)
ldohseek ..... ldohseek (S)
ldopen ..... ldopen (S)
ldrseek ..... ldrseek (S)
ldshread ..... ldshread (S)
ldsseek ..... ldsseek (S)
ldsysdump ..... ldsysdump (ADM)
ldtbindex ..... ldtbindex (S)
ldtbread ..... ldtbread (S)
ldtbseek ..... ldtbseek (S)
leaveok ..... curses (S)
leaveok ..... tam (S)
leaveok ..... terminfo (S)
lex ..... lex (CP)
lf ..... ls (C)
lfind ..... lsearch (S)
lgamma ..... gamma (S)
libwindows ..... libwindows (S)
limits ..... limits (FP)
line ..... line (C)
line ..... plot (S)
linemod ..... plot (S)
linenum ..... linenum (FP)
link ..... link (ADM)
link ..... link (S)
link_field ..... field (S)
link_fieldtype ..... fieldtype (S)
link_unix ..... link_unix (ADM)
lint ..... lint (CP)
list ..... list (ADM)
list ..... list (CP)
ll_close ..... llog (S)
ll_err ..... llog (S)
ll_hdinit ..... llog (S)
ll_init ..... llog (S)
ll_log ..... llog (S)
llog ..... llog (S)
ll_open ..... llog (S)
ln ..... ln (C)
__loc1 ..... regcmp (S)
loc1 ..... regexp (S)
loc2 ..... regexp (S)
locale ..... locale (C)
locale ..... locale (M)
localeconv ..... localeconv (S)
localedef ..... localedef (C)
localtime ..... ctime (S)
lock ..... lock (C)
lock ..... lock (S)
locked_out ..... fields (S)
lockf ..... lockf (S)
locking ..... locking (S)
locs ..... regexp (S)
log ..... exp (S)
log ..... log (HW)
log ..... log (M)
log10 ..... exp (S)
login ..... login (M)
logname ..... logname (C)
logname ..... logname (S)
logs ..... logs (F)
longjmp ..... setjmp (S)
longname ..... curses (S)
longname ..... terminfo (S)
lorder ..... lorder (CP)
lp ..... lp (C)

lp ..... lp (HW)
lp0 ..... lp (HW)
lp1 ..... lp (HW)
lp2 ..... lp (HW)
lpadmin ..... lpadmin (ADM)
lpfilter ..... lpfilter (ADM)
lpforms ..... lpforms (ADM)
lpmove ..... lpmove (ADM)
lpr ..... lp (C)
lprint ..... lprint (C)
lprof ..... lprof (CP)
lpsched ..... lpsched (ADM)
lpsh ..... lpsh (ADM)
lpshut ..... lpsched (ADM)
lpstat ..... lpstat (C)
lpusers ..... lpusers (ADM)
lr ..... ls (C)
lrand48 ..... drand48 (S)
ls ..... ls (C)
lsearch ..... lsearch (S)
lseek ..... lseek (S)
lstat ..... stat (S)
ltol3 ..... l3tol (S)
lx ..... ls (C)
m4 ..... m4 (CP)
machid ..... machid (C)
mail ..... mail (C)
maildelivery ..... maildelivery (F)
mailx ..... mail (C)
majorsinuse ..... majorsinuse (ADM)
make ..... make (CP)
makekey ..... makekey (ADM)
make_transition_files ..... dblock (S)
mallinfo ..... mallinfo (FP)
mallinfo ..... malloc (S)
malloc ..... malloc (S)
mallopt ..... malloc (S)
man ..... man (C)
mapchan ..... mapchan (F)
mapchan ..... mapchan (M)
mapkey ..... mapkey (M)
mapscrn ..... mapkey (M)
mapstr ..... mapkey (M)
mar ..... mar (CP)
masm ..... masm (CP)
math ..... math (M)
matherr ..... matherr (S)
maxuuscheds ..... maxuuscheds (F)
maxuuxqts ..... maxuuxqts (F)
mblen ..... mblen (S)
mbstowcs ..... mblen (S)
mbtowc ..... mblen (S)
mc68k ..... machid (C)
mcart ..... mcart (C)
mcconfig ..... mcconfig (F)
mcdaemon ..... mcconfig (F)
mcs ..... mcs (CP)
mdevice ..... mdevice (F)
mem ..... mem (FP)
memccpy ..... memory (S)
memchr ..... memory (S)
memcmp ..... memory (S)
memcpy ..... memory (S)
memmove ..... memmove (S)
memory ..... memory (S)
memset ..... memory (S)
memsize ..... memsize (ADM)
menu ..... menu (S)
menu ..... tam (S)
menu_add ..... undocumented (M)
menu_back ..... menu (S)
menu_del ..... undocumented (M)
menu_driver ..... menu (S)
menu_fore ..... menu (S)
menu_format ..... menu (S)
menu_grey ..... menu (S)
menu_init ..... menu (S)
menu_items ..... menu (S)
menu_mark ..... menu (S)
menumerge ..... menumerge (ADM)
menu_opts ..... menu (S)
menu_opts_off ..... menu (S)
menu_opts_on ..... menu (S)
menu_pad ..... menu (S)
menu_pattern ..... menu (S)
menu_sub ..... menu (S)
menu_term ..... menu (S)
menu_userptr ..... menu (S)
menu_win ..... menu (S)
mesg ..... mesg (C)
message ..... tam (S)
message ..... undocumented (M)
messages ..... messages (M)
mestbl ..... mestbl (M)
meta ..... curses (S)
meta ..... terminfo (S)
mfsys ..... mfsys (FP)
mkdev ..... mkdev (ADM)
mkdir ..... mkdir (C)
mkdir ..... mkdir (S)
mkfifo ..... mkfifo (C)
mkfifo ..... mkfifo (S)

mkfs ..... mkfs (ADM)
mknod ..... mknod (C)
mknod ..... mknod (S)
mkshlib ..... mkshlib (CP)
mkstr ..... mkstr(CP)
mktemp ..... mktemp (S)
mktime ..... mktime (S)
ml_1adr ..... ml_send (S)
ml_adr ..... ml_send (S)
ml_aend ..... ml_send (S)
ml_cc ..... ml_send (S)
ml_end ..... ml_send (S)
ml_file ..... ml_send (S)
ml_init ..... ml_send (S)
mlist ..... undocumented (M)
ml_send ..... ml_send (S)
ml_tinit ..... ml_send (S)
ml_to ..... ml_send (S)
ml_txt ..... ml_send (S)
mmdf ..... mmdf (ADM)
mmdf ..... mmdf (S)
mmdfalias ..... mmdfalias (ADM)
mmdftailor ..... mmdftailor(F)
mm_end ..... mmdf (S)
mm_init ..... mmdf (S)
mm_pkend ..... mmdf (S)
mm_pkinit ..... mmdf(S)
mm_radr ..... mmdf (S)
mm_rinit ..... mmdf (S)
mm_rrec ..... mmdf (S)
mm_rrply ..... mmdf(S)
mm_rstm ..... mmdf (S)
mm_rtxt ..... mmdf (S)
mm_sbend ..... mmdf (S)
mm_sbinit ..... mmdf(S)
mm_wadr ..... mmdf (S)
mm_waend ..... mmdf (S)
mm_winit ..... mmdf (S)
mm_wrec ..... mmdf(S)
mm_wrply ..... mmdf(S)
mm_wstm ..... mmdf(S)
mm_wtend ..... mmdf (S)
mm_wtxt ..... mmdf (S)
mnlist ..... mnlist (ADM)
mnt ..... mnt(C)
mnttab ..... mnttab (F)
modf ..... frexp (S)
monacct ..... acctsh (ADM)
monitor ..... monitor (S)
montbl ..... montbl (M)
more ..... more(C)
mount ..... mount (ADM)
mount ..... mount (S)
mountall ..... mountall (ADM)
mouse ..... mouse (HW)
move ..... curses (S)
Move ..... libwindows (S)
move ..... plot (S)
move ..... tam (S)
move ..... terminfo (S)
move_field ..... field (S)
move_panel ..... panel (S)
mrand48 ..... drand48 (S)
mscreen ..... mscreen (M)
mscsi ..... mscsi (F)
msg ..... msg (FP)
msgctl ..... msgctl (S)
msgget ..... msgget (S)
msgop ..... msgop (S)
msgrcv ..... msgop (S)
msgsnd ..... msgop (S)
mt ..... undocumented (M)
mtune ..... mtune(F)
multiscreen ..... multiscreen (M)
mv ..... mv (C)
mvaddch ..... curses (S)
mvaddch ..... tam (S)
mvaddch ..... terminfo (S)
mvaddstr ..... curses (S)
mvaddstr ..... tam (S)
mvaddstr ..... terminfo (S)
mvcur ..... curses(S)
mvcur ..... terminfo (S)
mvdelch ..... curses (S)
mvdelch ..... terminfo (S)
mvdevice ..... mvdevice(F)
mvdir ..... mvdir (ADM)
mvgetch ..... curses (S)
mvgetch ..... terminfo (S)
mvgetstr ..... curses(S)
mvgetstr ..... terminfo (S)
mvinch ..... curses (S)
mvinch ..... tam (S)
mvinch ..... terminfo (S)
mvinsch ..... curses (S)
mvinsch ..... terminfo (S)
mvprintw ..... curses (S)
mvprintw ..... terminfo (S)
mvscanw ..... curses (S)
mvscanw ..... terminfo (S)
mvwaddch ..... curses (S)
mvwaddch ..... terminfo (S)

mvwaddstr ..... curses (S)
mvwaddstr ..... terminfo (S)
mvwdelch ..... curses (S)
mvwdelch ..... terminfo (S)
mvwgetch ..... curses (S)
mvwgetch ..... terminfo (S)
mvwgetstr ..... curses (S)
mvwgetstr ..... terminfo (S)
mvwin ..... curses (S)
mvwin ..... terminfo (S)
mvwinch ..... curses (S)
mvwinch ..... terminfo (S)
mvwinsch ..... curses (S)
mvwinsch ..... terminfo (S)
mvwprintw ..... curses (S)
mvwprintw ..... terminfo (S)
mvwscanw ..... curses (S)
mvwscanw ..... terminfo (S)
nap ..... nap (S)
napms ..... curses (S)
napms ..... terminfo (S)
nawk ..... awk (C)
nbra ..... regexp (S)
nbwaitsem ..... waitsem (S)
ncheck ..... ncheck (ADM)
netbuf ..... netbuf (FP)
netconfig ..... netconfig (ADM)
netutil ..... netutil (ADM)
New ..... libwindows (S)
new_field ..... field (S)
new_fieldtype ..... fieldtype (S)
new_form ..... form (S)
newform ..... newform (C)
newgrp ..... newgrp (C)
new_item ..... item (S)
Newlayer ..... libwindows (S)
newmail ..... undocumented (M)
new_menu ..... menu (S)
newpad ..... curses (S)
newpad ..... terminfo (S)
new_page ..... form (S)
new_panel ..... panel (S)
news ..... news (C)
newterm ..... curses (S)
newterm ..... terminfo (S)
newwin ..... curses (S)
newwin ..... terminfo (S)
_nextchoice ..... fieldtype (S)
nextkey ..... dbm (S)
nice ..... nice (C)
nice ..... nice(S)
nictable ..... nictable (ADM)
nl ..... curses (S)
nl ..... nl (C)
nl ..... tam (S)
nl ..... terminfo (S)
nl_ascxtime ..... nl_cxtime (S)
nl_cxtime ..... nl_cxtime (S)
nl_fprintf ..... nl_printf (S)
nl_fscanf ..... nl_scanf (S)
nl_init ..... nl_init (S)
nlist ..... nlist (S)
nl_langinfo ..... nl_langinfo (S)
nl_printf ..... nl_printf (S)
nlsadmin ..... nlsadmin (ADM)
nl_scanf ..... nl_scanf (S)
nl_sprintf ..... nl_printf (S)
nl_sscanf ..... nl_scanf (S)
nl_strcmp ..... nl_strcmp (S)
nl_strncmp ..... nl_strcmp (S)
nl_types ..... nl_types (FP)
nm ..... nm (CP)
nm ..... nm (XNX)
nocbreak ..... curses (S)
nocbreak ..... tam (S)
nocbreak ..... terminfo (S)
nocrmode ..... curses(S)
nocrmode ..... tam (S)
nocrmode ..... terminfo (S)
nodelay ..... curses (S)
nodelay ..... tam (S)
nodelay ..... terminfo (S)
noecho ..... curses (S)
noecho ..... tam (S)
noecho ..... terminfo (S)
nohup ..... nohup (C)
nonl ..... curses (S)
nonl ..... tam (S)
nonl ..... terminfo (S)
noraw ..... curses (S)
noraw ..... terminfo (S)
notimeout ..... curses (S)
notimeout ..... terminfo (S)
nrand48 ..... drand48 (S)
nssend ..... nssend (FP)
null ..... null (F)
null ..... null (FP)
nulladm ..... acctsh (ADM)
numtbl ..... numtbl (M)
oawk ..... awk (C)
od ..... od (C)
open ..... open (S)

openagent ..... libwindows(S)
openchan ..... libwindows(S)
opendir ..... directory(S)
openpl ..... plot(S)
opensem ..... opensem(S)
optarg ..... getopt(S)
opterr ..... getopt(S)
optind ..... getopt(S)
os2ld ..... os2ld(CP)
ospeed ..... termcap(S)
overlay ..... curses(S)
overlay ..... terminfo(S)
overwrite ..... curses(S)
overwrite ..... terminfo(S)
paccess ..... paccess(S)
pack ..... pack(C)
page ..... more(C)
pair_content ..... curses(S)
pair_content ..... terminfo(S)
panel ..... panel(S)
panel_above ..... panel(S)
panel_below ..... panel(S)
panel_hidden ..... panel(S)
panel_userptr ..... panel(S)
panel_window ..... panel(S)
parallel ..... parallel(HW)
passlen ..... passlen(S)
passwd ..... passwd(C)
passwd ..... passwd(F)
passwd ..... passwd(FP)
paste ..... paste(C)
pathchk ..... pathchk(C)
pathconf ..... pathconf(S)
pause ..... pause(S)
pax ..... pax(C)
pb_check ..... tam(S)
pb_empty ..... tam(S)
pb_gbuf ..... tam(S)
pb_gets ..... tam(S)
pb_name ..... tam(S)
pb_open ..... tam(S)
pb_puts ..... tam(S)
pb_seek ..... tam(S)
pb_weof ..... tam(S)
PC ..... termcap(S)
pcat ..... pack(C)
pclose ..... popen(S)
pcpio ..... pcpio(C)
pdp11 ..... machid(C)
pechochar ..... curses(S)
pechochar ..... terminfo(S)
PEEKC ..... regexp(S)
permissions ..... permissions(F)
perms ..... perms(F)
perror ..... perror(S)
pg ..... pg(C)
phs ..... phs(S)
phs_get ..... phs(S)
phs_msg ..... phs(S)
phs_note ..... phs(S)
pipe ..... pipe(ADM)
pipe ..... pipe(S)
pkgadd ..... pkgadd(ADM)
pkgask ..... pkgask(ADM)
pkgchk ..... pkgchk(ADM)
pkginfo ..... pkginfo(ADM)
pkginfo ..... pkginfo(F)
pkgmap ..... pkgmap(F)
pkgmk ..... pkgmk(ADM)
pkgparam ..... pkgparam(ADM)
pkgproto ..... pkgproto(ADM)
pkgrm ..... pkgrm(ADM)
pkgtrans ..... pkgtrans(ADM)
plock ..... plock(S)
plot ..... plot(FP)
plot ..... plot(S)
pnch ..... pnch(FP)
pnoutrefresh ..... curses(S)
pnoutrefresh ..... terminfo(S)
point ..... plot(S)
Poll ..... poll(F)
poll ..... poll(S)
Poll.day ..... poll(F)
Poll.hour ..... poll(F)
popen ..... popen(S)
pos_form_cursor ..... form(S)
pos_menu_cursor ..... menu(S)
post_form ..... form(S)
post_menu ..... menu(S)
pow ..... exp(S)
pr ..... pr(C)
prctmp ..... acctsh(ADM)
prdaily ..... acctsh(ADM)
prefresh ..... curses(S)
prefresh ..... terminfo(S)
_prevchoice ..... fieldtype(S)
prf ..... prf(HW)
prfdc ..... profiler(ADM)
prfld ..... profiler(ADM)
prfpr ..... profiler(ADM)
prfsnap ..... profiler(ADM)
prfstat ..... profiler(ADM)

primary_auth ..... subsystems(S)
primary_of_secondary_auth ..... subsystems(S)
printenv ..... env(C)
printf ..... printf(S)
printw ..... curses(S)
printw ..... tam(S)
printw ..... terminfo(S)
proc ..... proc(FP)
proctl ..... proctl(S)
prof ..... prof(CP)
prof ..... prof(M)
prof ..... prof(XNX)
profil ..... profil(S)
profile ..... profile(M)
profiler ..... profiler(ADM)
proto ..... proto(F)
prototype ..... prototype(F)
prpw ..... prpw(F)
prs ..... prs(CP)
prtacct ..... acctsh(ADM)
prwarn ..... prwarn(C)
ps ..... ps(C)
pstat ..... pstat(C)
ptar ..... ptar(C)
ptmx ..... ptmx(M)
ptrace ..... ptrace(S)
pts??? ..... ptmx(M)
purge ..... purge(C)
purge ..... purge(F)
putc ..... putc(S)
putchar ..... putc(S)
putdvagnam ..... getdvagent(S)
putenv ..... putenv(S)
putmsg ..... putmsg(S)
putp ..... curses(S)
putp ..... terminfo(S)
putprdfnam ..... getprdfent(S)
putprfinam ..... getprfient(S)
putprpwnam ..... getprpwent(S)
putprtcnam ..... getprtcent(S)
putpwent ..... putpwent(S)
puts ..... puts(S)
putspent ..... putspent(S)
pututline ..... getut(S)
putw ..... putc(S)
pwck ..... pwck(ADM)
pwconv ..... pwconv(ADM)
pwd ..... pwd(C)
pwdmenu ..... undocumented(M)
pw_idtoname ..... pw_nametoid(S)
pw_nametoid ..... pw_nametoid(S)
pwunconv ..... pwconv(ADM)
qsort ..... qsort(S)
queue ..... queue(F)
queuedefs ..... queuedefs(F)
quot ..... quot(C)
raise ..... raise(S)
ramdisk ..... ramdisk(HW)
rand ..... rand(S)
random ..... random(C)
randomword ..... randomword(S)
ranlib ..... ranlib(XNX)
raw ..... curses(S)
raw ..... terminfo(S)
rc0 ..... rc0(ADM)
rc2 ..... rc2(ADM)
rcc ..... rcc(CP)
rcflow ..... rcflow(CP)
rcp ..... rcp(C)
rcvalert ..... rcvalert(C)
rcvfile ..... rcvfile(C)
rcvprint ..... rcvprint(C)
rcvtrip ..... rcvtrip(C)
rcxref ..... rcxref(CP)
rdchk ..... rdchk(S)
read ..... read(S)
readdir ..... directory(S)
readlink ..... readlink(S)
read_pw_fields ..... fields(S)
read_tc_fields ..... fields(S)
realloc ..... malloc(S)
reboot ..... haltsys(ADM)
red ..... ed(C)
reduce ..... reduce(ADM)
refresh ..... curses(S)
refresh ..... tam(S)
refresh ..... terminfo(S)
regcmp ..... regcmp(CP)
regcmp ..... regcmp(S)
regex ..... regcmp(S)
regexp ..... regexp(S)
reject ..... accept(ADM)
relax ..... relax(ADM)
reloc ..... reloc(FP)
relogin ..... relogin(ADM)
remote ..... remote(C)
remove ..... remove(S)
removef ..... removef(ADM)
removepkg ..... removepkg(ADM)
rename ..... rename(S)
repackman ..... repackman(ADM)

replace_file ..... dblock (S)
replace_panel ..... panel (S)
resend ..... resend (C)
reset_prog_mode ..... curses (S)
reset_prog_mode ..... terminfo (S)
reset_shell_mode ..... curses (S)
reset_shell_mode ..... terminfo (S)
resetterm ..... curses (S)
reset_tty ..... curses (S)
reset_tty ..... tam (S)
reset_tty ..... terminfo (S)
resetty ..... curses (S)
resetty ..... tam (S)
resetty ..... terminfo (S)
Reshape ..... libwindows (S)
restartterm ..... curses (S)
restartterm ..... terminfo (S)
restore ..... restore (ADM)
RETURN ..... regexp (S)
rewind ..... fseek (S)
rewinddir ..... directory (S)
ripoffline ..... curses (S)
ripoffline ..... terminfo (S)
rksh ..... ksh (C)
rlint ..... rlint (CP)
rlpcmd ..... rlpcmd (C)
rlpstat ..... lpstat (C)
rm ..... rm (C)
rmail ..... rmail (ADM)
rmb ..... rmb (M)
rmdel ..... rmdel (CP)
rmdir ..... rmdir (C)
rmdir ..... rmdir (S)
rmgroup ..... rmuser (ADM)
rmpasswd ..... rmuser (ADM)
rmuser ..... rmuser (ADM)
Routines ..... Routines (DOS)
Routines ..... Routines (S)
rsh ..... rsh (C)
rstab ..... undocumented (M)
rtc ..... rtc (HW)
runacct ..... acctsh (ADM)
runacct ..... runacct (ADM)
run_crypt ..... crypt (S)
Runlayer ..... libwindows (S)
run_setkey ..... crypt (S)
sa1 ..... sar (ADM)
sa2 ..... sar (ADM)
sact ..... sact (CP)
sadc ..... sar (ADM)
sag ..... sag (ADM)
sar ..... sar (ADM)
saveterm ..... curses (S)
savetty ..... curses (S)
savetty ..... tam (S)
savetty ..... terminfo (S)
sbrk ..... brk (S)
scale_form ..... form (S)
scale_menu ..... menu (S)
scancode ..... scancode (HW)
scanf ..... scanf (S)
scanoff ..... scanon (M)
scanon ..... scanon (M)
scanw ..... curses (S)
scanw ..... terminfo (S)
sc_copyscstate ..... sc_raw (S)
sccsdiff ..... sccsdiff (CP)
sccsfile ..... sccsfile (FP)
sc_exit ..... sc_init (S)
sc_getfkeystr ..... sc_init (S)
sc_getinfo ..... sc_raw (S)
sc_getkbmap ..... sc_init (S)
sc_getkeymap ..... sc_init (S)
sc_getled ..... sc_init (S)
sc_getscreenswitch ..... sc_raw (S)
schedule ..... schedule (ADM)
sc_init ..... sc_init (S)
sc_kb2mapcode ..... sc_readkb (S)
sc_mapcode2kb ..... sc_readkb (S)
sc_mapcode2str ..... sc_readkb (S)
sc_mapin ..... sc_readkb (S)
sc_mapinit ..... sc_init (S)
sc_mapout ..... sc_readkb (S)
scnhdr ..... scnhdr (FP)
__scoinfo ..... __scoinfo (S)
scosh ..... scosh (C)
sc_raw ..... sc_raw (S)
scr_dump ..... curses (S)
scr_dump ..... scr_dump (FP)
scr_dump ..... terminfo (S)
sc_readkb ..... sc_readkb (S)
sc_readmapcode ..... sc_readkb (S)
sc_readstr ..... sc_readkb (S)
sc_receive_kb ..... sc_init (S)
screen ..... screen (HW)
scr_init ..... curses (S)
scr_init ..... terminfo (S)
scroll ..... curses (S)
scroll ..... terminfo (S)
scrollok ..... curses (S)
scrollok ..... terminfo (S)
scr_restore ..... curses (S)

scr_restore ..... terminfo (S)
sc_setfkeystr ..... sc_init (S)
sc_setinfo ..... sc_raw (S)
sc_setkeymap ..... sc_init (S)
sc_setled ..... sc_init (S)
sc_setscreenswitch ..... sc_raw (S)
scsi ..... scsi (HW)
scsibadblk ..... scsibadblk (ADM)
sc_str2kb ..... sc_readkb (S)
sc_unraw ..... sc_raw (S)
sd ..... sd (ADM)
sdb ..... sdb (CP)
sdd ..... sd (ADM)
sddate ..... sddate (C)
sdenter ..... sdenter (S)
sdevice ..... sdevice (F)
sdfree ..... sdget (S)
sdget ..... sdget (S)
sdgetv ..... sdgetv (S)
sdiff ..... sdiff (C)
sdleave ..... sdenter (S)
sdwaitv ..... sdgetv (S)
secondary_auth ..... subsystems (S)
sed ..... regexp (S)
sed ..... sed (C)
seed ..... seed (S)
seed48 ..... drand48 (S)
seekdir ..... directory (S)
select ..... select (S)
sem ..... sem (FP)
semctl ..... semctl (S)
semget ..... semget (S)
semop ..... semop (S)
serial ..... serial (HW)
set_auth_parameters ..... identity (S)
setbuf ..... setbuf (S)
setclk ..... undocumented (M)
setclock ..... setclock (ADM)
setcolor ..... setcolor (C)
setcolour ..... setcolor (C)
set_current_field ..... form (S)
set_current_item ..... menu (S)
set_curterm ..... curses (S)
set_curterm ..... terminfo (S)
setdvagent ..... getdvagent (S)
set_field_back ..... field (S)
set_field_buffer ..... field (S)
set_field_fore ..... field (S)
set_field_init ..... form (S)
set_field_just ..... field (S)
set_field_opts ..... field (S)
set_field_pad ..... field (S)
set_field_status ..... field (S)
set_field_term ..... form (S)
set_field_type ..... field (S)
set_fieldtype_arg ..... fieldtype (S)
set_fieldtype_choice ..... fieldtype (S)
set_field_userptr ..... field (S)
set_form_fields ..... form (S)
set_form_init ..... form (S)
set_form_opts ..... form (S)
set_form_page ..... form (S)
set_form_sub ..... form (S)
set_form_term ..... form (S)
set_form_userptr ..... form (S)
set_form_win ..... form (S)
setgid ..... setuid (S)
setgrent ..... getgrent (S)
setgroups ..... setgroups (S)
set_item_init ..... menu (S)
set_item_opts ..... item (S)
set_item_term ..... menu (S)
set_item_userptr ..... item (S)
set_item_value ..... item (S)
setitimer ..... getitimer (S)
setjmp ..... setjmp (S)
setkey ..... crypt (S)
setkey ..... setkey (C)
setlocale ..... setlocale (S)
setluid ..... setluid (S)
set_menu_back ..... menu (S)
set_menu_fore ..... menu (S)
set_menu_format ..... menu (S)
set_menu_grey ..... menu (S)
set_menu_init ..... menu (S)
set_menu_items ..... menu (S)
set_menu_mark ..... menu (S)
set_menu_opts ..... menu (S)
set_menu_pad ..... menu (S)
set_menu_pattern ..... menu (S)
set_menu_sub ..... menu (S)
set_menu_term ..... menu (S)
set_menu_userptr ..... menu (S)
set_menu_win ..... menu (S)
setmnt ..... setmnt (ADM)
set_new_page ..... form (S)
set_panel_userptr ..... panel (S)
setpgid ..... setpgid (S)
setpgrp ..... setpgrp (S)
setprdfent ..... getprdfent (S)
setprfient ..... getprfient (S)
setpriv ..... setpriv (S)

setprpwent ..... getprpwent (S)
setprtcent ..... getprtcent (S)
setpwent ..... getpwent (S)
setscrreg ..... curses (S)
setscrreg ..... terminfo (S)
set_seed ..... seed (S)
setsid ..... setsid (S)
setspent ..... getspent (S)
setsyx ..... curses (S)
setsyx ..... terminfo (S)
set_term ..... curses (S)
set_term ..... terminfo (S)
settime ..... settime (ADM)
set_top_row ..... menu (S)
set_tty ..... curses (S)
set_tty ..... terminfo (S)
setuid ..... setuid (S)
setupterm ..... curses (S)
setupterm ..... terminfo (S)
setutent ..... getut (S)
setvbuf ..... setbuf (S)
sfmt ..... sfmt (ADM)
sfsys ..... sfsys (FP)
sg ..... sg (C)
sgetl ..... sputl (S)
sh ..... sh(C)
shadow ..... shadow (F)
shl ..... shl (C)
shm ..... shm (FP)
shmat ..... shmop (S)
shmctl ..... shmctl (S)
shmdt ..... shmop (S)
shmget ..... shmget (S)
shmop ..... shmop (S)
show_panel ..... panel (S)
shutacct ..... acctsh (ADM)
shutdn ..... shutdn (S)
shutdown ..... shutdown (ADM)
sigaction ..... sigaction (S)
sigaddset ..... sigset (S)
sigdelset ..... sigset (S)
sigemptyset ..... sigset (S)
sigfillset ..... sigset (S)
sighold ..... sigsetv (S)
sigignore ..... sigsetv (S)
sigismember ..... sigset (S)
siglongjmp ..... sigsetjmp (S)
signal ..... signal (S)
signgam ..... gamma (S)
sigpause ..... sigsetv (S)
sigpending ..... sigpending (S)
sigprocmask ..... sigprocmask (S)
sigrelse ..... sigsetv (S)
sigsem ..... sigsem (S)
sigset ..... sigset (S)
sigset ..... sigsetv (S)
sigsetjmp ..... sigsetjmp (S)
sigsuspend ..... sigsuspend (S)
sin ..... trig (S)
sinh ..... sinh (S)
S_ISBLK ..... stat (S)
S_ISCHR ..... stat (S)
S_ISDIR ..... stat (S)
S_ISFIFO ..... stat (S)
S_ISNAM ..... stat (S)
S_ISREG ..... stat (S)
size ..... size (CP)
size ..... size (XNX)
sleep ..... sleep (C)
sleep ..... sleep (S)
slk_clear ..... curses (S)
slk_clear ..... terminfo (S)
slk_init ..... curses (S)
slk_init ..... terminfo (S)
slk_label ..... curses (S)
slk_label ..... terminfo (S)
slk_noutrefresh ..... curses (S)
slk_noutrefresh ..... terminfo (S)
slk_refresh ..... curses (S)
slk_refresh ..... terminfo (S)
slk_restore ..... curses (S)
slk_restore ..... terminfo (S)
slk_set ..... curses (S)
slk_set ..... terminfo (S)
slk_touch ..... curses (S)
slk_touch ..... terminfo (S)
slot ..... slot (C)
smmck ..... tcbck (ADM)
sort ..... sort (C)
space ..... plot (S)
space ..... space (F)
spell ..... spell (C)
spellin ..... spell (C)
spline ..... spline (C)
split ..... split(C)
sprintf ..... printf (S)
sputl ..... sputl (S)
sqrt ..... exp (S)
srand ..... rand (S)
srand48 ..... drand48 (S)
sscanf ..... scanf(S)
ssignal ..... ssignal (S)

standend ..... curses (S)
standend ..... terminfo (S)
standout ..... curses (S)
standout ..... terminfo (S)
start_color ..... curses (S)
start_color ..... terminfo (S)
starting_egid ..... identity (S)
starting_euid ..... identity (S)
starting_luid ..... identity (S)
starting_rgid ..... identity (S)
starting_ruid ..... identity (S)
startup ..... acctsh (ADM)
stat ..... stat (FP)
stat ..... stat (S)
statfs ..... statfs(S)
statlstat ..... stat (S)
stdarg ..... varargs (S)
stdbl ..... isconv (S)
stderr ..... stdio (S)
stdin ..... stdio (S)
stdio ..... stdio (S)
stdout ..... stdio (S)
step ..... regexp (S)
stfloat ..... isconv (S)
stime ..... stime (S)
stint ..... isconv (S)
stlong ..... isconv (S)
stopio ..... stopio (S)
store ..... dbm (S)
store_pw_fields ..... fields(S)
store_tc_fields ..... fields (S)
strace ..... strace (ADM)
strcat ..... string (S)
strchr ..... string (S)
strclean ..... strclean (ADM)
strcmp ..... string (S)
strcoll ..... strcoll (S)
strcpy ..... string (S)
strcspn ..... string (S)
strdup ..... string (S)
streamio ..... streamio (M)
strerr ..... strerr(ADM)
strerror ..... strerror (S)
strftime ..... ctime (S)
strftime ..... strftime (S)
string ..... string (M)
string ..... string (S)
strings ..... strings (C)
strip ..... strip (CP)
strip ..... strip (XNX)
strlen ..... string (S)
strncat ..... string (S)
strncmp ..... string (S)
strncoll ..... strcoll (S)
strncpy ..... string (S)
strnxfrm ..... strcoll (S)
strpbrk ..... string (S)
strrchr ..... string (S)
strspn ..... string (S)
strstr ..... string (S)
strtod ..... strtod (S)
strtok ..... string (S)
strtol ..... strtol (S)
strtoul ..... strtoul (S)
strxfrm ..... strcoll (S)
STTY ..... stty (C)
stune ..... stune (F)
su ..... su (C)
submit ..... submit (ADM)
subpad ..... curses (S)
subpad ..... terminfo (S)
subsystem ..... subsystem (M)
subsystems ..... subsystems (S)
subwin ..... curses (S)
subwin ..... terminfo (S)
sulogin ..... sulogin (ADM)
sum ..... sum (C)
swab ..... swab (S)
swap ..... swap (ADM)
swconfig ..... swconfig (C)
sxt ..... sxt (M)
symlink ..... symlink (S)
syms ..... syms (FP)
sync ..... sync (ADM)
sync ..... sync (S)
sysadmcolor ..... sysadmcolor (F)
sysadmmenu ..... sysadmmenu (F)
sysadmsh ..... sysadmsh (ADM)
sysconf ..... sysconf (S)
sysdef ..... sysdef (ADM)
sys_errlist ..... perror (S)
sysfiles ..... sysfiles (F)
sysfs ..... sysfs (S)
sysi86 ..... sysi86 (S)
sys_nerr ..... perror (S)
system ..... system (S)
systemid ..... systemid (F)
systems ..... systems (F)
systty ..... systty (M)
tables ..... tables (F)
tabs ..... tabs (C)
t_accept ..... t_accept (S)

tai_end ..... tai (S)
tai_get ..... tai (S)
tai_init ..... tai (S)
tail ..... tail (C)
t_alloc ..... t_alloc (S)
tam ..... tam (S)
tan ..... trig (S)
tanh ..... sinh (S)
tape ..... tape (C)
tape ..... tape (HW)
tapecntl ..... tapecntl (C)
tapedump ..... tapedump (C)
tar ..... tar (C)
tar ..... tar (F)
t_bind ..... t_bind (S)
tcbck ..... tcbck (ADM)
tcdrain ..... tcflow (S)
tcflow ..... tcflow (S)
tcflush ..... tcflow (S)
tcgetattr ..... tcattr (S)
tcgetpgrp ..... tcpgrp (S)
tcl ..... tcl (C)
t_close ..... t_close (S)
t_connect ..... t_connect (S)
tcsendbreak ..... tcflow (S)
tcsetattr ..... tcattr(S)
tcsetpgrp ..... tcpgrp (S)
tdelete ..... tsearch (S)
tee ..... tee (C)
telinit ..... init (M)
telldir ..... directory (S)
tempnam ..... tmpnam (S)
term ..... term (M)
termcap ..... termcap (F)
termcap ..... termcap (S)
terminal ..... terminal (HW)
terminals ..... terminals (M)
terminfo ..... terminfo (F)
terminfo ..... terminfo (M)
terminfo ..... terminfo (S)
termio ..... termio (M)
termios ..... termios (M)
termupd ..... ttyupd (ADM)
t_errlist ..... t_error (S)
t_errno ..... t_error (S)
t_error ..... t_error (S)
test ..... test (C)
tfind ..... tsearch (S)
t_free ..... t_free(S)
tgetent ..... curses (S)
tgetent ..... termcap (S)
tgetent ..... terminfo (S)
tgetflag ..... curses (S)
tgetflag ..... termcap (S)
tgetflag ..... terminfo (S)
t_getinfo ..... t_getinfo (S)
tgetnum ..... curses (S)
tgetnum ..... termcap (S)
tgetnum ..... terminfo (S)
t_getstate ..... t_getstate (S)
tgetstr ..... curses (S)
tgetstr ..... termcap (S)
tgetstr ..... terminfo (S)
tgoto ..... curses (S)
tgoto ..... termcap (S)
tgoto ..... terminfo (S)
tic ..... tic (C)
tigetflag ..... curses (S)
tigetflag ..... terminfo (S)
tigetnum ..... curses (S)
tigetnum ..... terminfo (S)
tigetstr ..... curses (S)
tigetstr ..... terminfo (S)
time ..... time (C)
time ..... time (S)
times ..... times (S)
timex ..... timex (ADM)
timezone ..... ctime (S)
timezone ..... timezone (F)
timod ..... timod (M)
timtbl ..... timtbl (M)
t_info ..... t_info (FP)
tirdwr ..... tirdwr (M)
t_listen ..... t_listen (S)
t_look ..... t_look (S)
tmpfile ..... tmpfile (S)
tmpnam ..... tmpnam (S)
t_nerr ..... t_error (S)
toascii ..... ctype(S)
toascii ..... toascii (S)
todigit ..... toascii (S)
toint ..... toascii (S)
tolower ..... ctype (S)
_tolower ..... toascii (S)
Top ..... libwindows (S)
top ..... top (F)
t_open ..... t_open (S)
top.next ..... top (F)
top_panel ..... panel (S)
top_row ..... menu (S)
t_optmgmt ..... t_optmgmt (S)
total_auths ..... subsystems (S)

touch ..... touch (C)
touchline ..... curses (S)
touchline ..... terminfo (S)
touchwin ..... curses (S)
touchwin ..... terminfo (S)
_toupper ..... ctype (S)
_toupper ..... toascii (S)
tparm ..... curses (S)
tparm ..... terminfo (S)
tplot ..... tplot (ADM)
tput ..... tput (C)
tputs ..... curses (S)
tputs ..... termcap (S)
tputs ..... terminfo (S)
tr ..... tr (C)
traceoff ..... curses (S)
traceoff ..... terminfo (S)
traceon ..... curses (S)
traceon ..... terminfo (S)
track ..... tam (S)
translate ..... translate (C)
trchan ..... trchan (M)
t_rcv ..... t_rcv (S)
t_rcvconnect ..... t_rcvconnect (S)
t_rcvdis ..... t_rcvdis (S)
t_rcvrel ..... t_rcvrel (S)
t_rcvudata ..... t_rcvudata (S)
t_rcvuderr ..... t_rcvuderr (S)
trig ..... trig (S)
true ..... true (C)
tsearch ..... tsearch (S)
tset ..... tset (C)
t_snd ..... t_snd (S)
t_snddis ..... t_snddis (S)
t_sndrel ..... t_sndrel (S)
t_sndudata ..... t_sndudata (S)
tsort ..... tsort (CP)
t_sync ..... t_sync (S)
tty ..... tty (C)
tty ..... tty (M)
tty1[a-h] ..... serial (HW)
tty2[a-h] ..... serial (HW)
ttyname ..... ttyname (S)
ttys ..... ttys (F)
ttyslot ..... ttyslot (S)
ttytype ..... ttytype (F)
ttyupd ..... ttyupd (ADM)
t_unbind ..... t_unbind (S)
turnacct ..... acctsh (ADM)
twalk ..... tsearch (S)
typeahead ..... curses (S)
typeahead ..... terminfo (S)
types ..... types (FP)
tz ..... tz (M)
tzname ..... ctime (S)
tzset ..... ctime (S)
u370 ..... machid (C)
u3b ..... machid (C)
u3b15 ..... machid (C)
u3b2 ..... machid (C)
u3b5) ..... machid (C)
uadmin ..... uadmin (ADM)
uadmin ..... uadmin (S)
ulckpwdf ..... getspent (S)
ulimit ..... ulimit (S)
umask ..... umask (C)
umask ..... umask (S)
umnt ..... mnt (C)
umount ..... mount (ADM)
umount ..... umount (S)
umountall ..... mountall (ADM)
uname ..... uname (C)
uname ..... uname (S)
uncompress ..... compress (C)
unctrl ..... curses (S)
unctrl ..... terminfo (S)
undial ..... dial (S)
undocumented ..... undocumented (M)
unexecseg ..... execseg (S)
unexpand ..... expand (C)
unget ..... unget (CP)
UNGETC ..... regexp (S)
ungetc ..... ungetc (S)
ungetch ..... curses (S)
ungetch ..... terminfo (S)
uniq ..... uniq (C)
unistd ..... unistd (FP)
units ..... units (C)
unlink ..... link (ADM)
unlink ..... unlink (S)
unpack ..... pack (C)
unpost_form ..... form (S)
unpost_menu ..... menu (S)
unretire ..... unretire (ADM)
UP ..... termcap (S)
update ..... undocumented (M)
update_panels ..... panel (S)
uptime ..... uptime (C)
usemouse ..... usemouse (C)
ustat ..... ustat (S)
utime ..... utime (S)
utmp ..... utmp (F)

utmp_getty ..... undocumented (M)
utmpname ..... getut (S)
uuchat ..... dial (ADM)
uucheck ..... uucheck (ADM)
uucico ..... uucico (ADM)
uuclean ..... uuclean (ADM)
uucp ..... uucp (C)
uudecode ..... uuencode (C)
uudemon ..... uudemon (ADM)
uudemon.admin ..... uudemon (ADM)
uudemon.clean ..... uudemon (ADM)
uudemon.hour ..... uudemon (ADM)
uudemon.poll ..... uudemon (ADM)
uudemon.poll2 ..... uudemon (ADM)
uuencode ..... uuencode (C)
uugetty ..... getty (M)
uuinstall ..... uuinstall (ADM)
uulist ..... uulist (ADM)
uulog ..... uucp (C)
uuname ..... uucp (C)
uupick ..... uuto (C)
uusched ..... uusched (ADM)
uustat ..... uustat (C)
uuto ..... uuto (C)
uutry ..... uutry (ADM)
uux ..... uux (C)
uuxqt ..... uuxqt (ADM)
va_alist ..... varargs (S)
va_arg ..... varargs (S)
va_dcl ..... varargs (S)
va_end ..... varargs (S)
val ..... val (CP)
va_list ..... varargs (S)
values ..... values (M)
varargs ..... varargs (S)
va_start ..... varargs (S)
vax ..... machid (C)
vc ..... vc (CP)
vectorsinuse ..... vectorsinuse (ADM)
vedit ..... vi (C)
vfprintf ..... vprintf (S)
vi ..... vi (C)
vidattr ..... curses (S)
vidattr ..... terminfo (S)
vidi ..... vidi (C)
vidputs ..... curses (S)
vidputs ..... terminfo (S)
view ..... vi (C)
vldldptr ..... ldptr(S)
vmstat ..... vmstat (C)
volcopy ..... volcopy (ADM)
vprintf ..... vprintf (S)
vsprintf ..... vprintf (S)
vwprintw ..... curses (S)
vwprintw ..... terminfo (S)
vwscanw ..... curses (S)
vwscanw ..... terminfo (S)
w ..... w (C)
waddch ..... curses (S)
waddch ..... terminfo (S)
waddstr ..... curses (S)
waddstr ..... terminfo (S)
wait ..... wait (C)
wait ..... wait (S)
waitpid ..... wait (S)
waitsem ..... waitsem (S)
wall ..... wall (ADM)
wattroff ..... curses (S)
wattroff ..... terminfo (S)
wattron ..... curses (S)
wattron ..... terminfo (S)
wattrset ..... curses (S)
wattrset ..... terminfo (S)
wc ..... wc (C)
wclear ..... curses (S)
wclear ..... terminfo (S)
wclrtobot ..... curses (S)
wclrtobot ..... terminfo (S)
wclrtoeol ..... curses (S)
wclrtoeol ..... terminfo (S)
wcmd ..... tam (S)
wcreate ..... tam (S)
wcstombs ..... mblen (S)
wctomb ..... mblen (S)
wdelch ..... curses (S)
wdelch ..... terminfo (S)
wdelete ..... tam (S)
wdeleteln ..... curses (S)
wdeleteln ..... terminfo (S)
wechochar ..... curses (S)
wechochar ..... terminfo (S)
werase ..... curses (S)
werase ..... terminfo (S)
wexit ..... tam (S)
wgetc ..... tam (S)
wgetch ..... curses (S)
wgetch ..... terminfo (S)
wgetmouse ..... tam (S)
wgetpos ..... tam (S)
wgetsel ..... tam (S)
wgetstat ..... tam (S)
wgetstr ..... curses (S)

wgetstr ..... terminfo (S)
wgoto ..... tam (S)
what ..... what (C)
what ..... what (CP)
who ..... who (C)
whodo ..... whodo (C)
wicoff ..... tam (S)
wicon ..... tam (S)
widest_auth ..... subsystems (S)
winch ..... curses (S)
winch ..... terminfo (S)
wind ..... tam (S)
winit ..... tam (S)
winsch ..... curses (S)
winsch ..... terminfo (S)
winsertln ..... curses (S)
winsertln ..... terminfo (S)
wlabel ..... tam (S)
wmove ..... curses (S)
wmove ..... terminfo (S)
wndelay ..... tam (S)
wnl ..... tam (S)
wnoutrefresh ..... curses (S)
wnoutrefresh ..... terminfo (S)
wpostwait ..... tam (S)
wprexec ..... tam (S)
wprintf ..... tam (S)
wprintw ..... curses (S)
wprintw ..... terminfo (S)
wprompt ..... tam (S)
wputc ..... tam (S)
wputs ..... tam (S)
wrastop ..... tam (S)
wreadmouse ..... tam (S)
wrefresh ..... curses (S)
wrefresh ..... tam (S)
wrefresh ..... terminfo (S)
write ..... write (C)
write ..... write (S)
write_authorizations ..... subsystems (S)
wscanw ..... curses (S)
wscanw ..... terminfo (S)
wselect ..... tam (S)
wserver ..... wserver (C)
wsetmouse ..... tam (S)
wsetscrreg ..... curses (S)
wsetscrreg ..... terminfo (S)
wsetstat ..... tam (S)
wslk ..... tam (S)
wstandend ..... curses (S)
wstandend ..... terminfo (S)
wstandout ..... curses (S)
wstandout ..... terminfo (S)
wtinit ..... wtinit (ADM)
wtmp ..... utmp (F)
wtmpfix ..... fwtmp (ADM)
wuser ..... tam (S)
x286emul ..... x286emul (C)
x286emul ..... x286emul (CP)
xargs ..... xargs (C)
xbackup ..... xbackup (ADM)
xbackup ..... xbackup (F)
xdump ..... xbackup (ADM)
xdumpdir ..... xdumpdir (ADM)
xinstall ..... xinstall (ADM)
xlist ..... xlist (S)
x.out ..... x.out (FP)
xrestor ..... xrestore (ADM)
xrestore ..... xrestore (ADM)
xstr ..... xstr (CP)
xt ..... xt (HW)
xtd ..... xtd (ADM)
xtil ..... xtil (CP)
xtod ..... xtod (C)
xtproto ..... xtproto (M)
xtract ..... xtract (C)
xts ..... xts (ADM)
xtt ..... xtt (ADM)
y0 ..... bessel (S)
y1 ..... bessel (S)
yacc ..... yacc (CP)
yes ..... yes (C)
yn ..... bessel (S)
zcat ..... compress (C)

Commands for Programming (CP)

Intro

Introduces Development System commands

Description

This section describes use of the standalone commands available in the Development System. Each command is labeled with the letters CP to distinguish it from commands available in the Operating System and other commands within the Development System. This convention points users to the documentation set that contains information on a particular command. For example, the reference cc(CP) indicates a reference to a discussion of the cc command in this section; the letter " C " stands for "Command" and the letter " P " stands for "Programming".

The ADM (administration) section contains miscellaneous information, including a great deal of system maintenance information. Other reference sections related to the Development System include the M (miscellaneous) section, the S (system services) section, and the FP (programming file formats) section.

Syntax

Unless otherwise noted, commands described in the "Syntax" section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below. This syntax and the rules which describe it are not followed by all commands, but they indicate what is generally true.

name [*-option*...] [*cmdarg*...]

[ ]	Surround a syntactic element that is not required.
...	Indicates multiple occurrences
name	The command name
option	*letter [arg[,...]]* An ***option*** is always preceded by a "-".
letter	Any single letter
arg	A character string that follows a ***letter*** in an ***option.***
cmdarg	A path name, filename, any other argument not beginning with a dash (-), or a dash by itself.

Command syntax standard rules

- Command names are between two and nine characters long and must include only lowercase letters and digits
- Options must be one character long
- All options must be preceded by a dash
- Options with no arguments may be grouped after a single dash
- The first *arg* following an option must be preceded by white space
- *arg*s cannot be optional
- Groups of *arg*s following an option must either be separated by commas or be separated by white space and quoted (for example, **-o xxx,z,yy** or **-o "xxx z yy"**)
- All options must precede *cmdarg*s on the command line
- A dash may be used to indicate the end of the options
- The order of the options relative to one another should not matter
- The order of the *cmdarg*s may be significant for some commands
- A dash preceded and followed by white space should only be used to indicate standard input

Diagnostics

Upon termination, each command returns two status bytes, one supplied by the system and giving the cause for termination, and, in the case of "normal" termination, one supplied by the program (see **wait**(S) and **exit**(S)). The first byte is 0 for normal termination; the second is customarily 0 for successful execution and non-zero to indicate problems such as erroneous parameters, bad or inaccessible data. The second byte is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

Note

Not all commands adhere to the syntax described here.

See also

getopts(C), **exit**(S), **getopt**(S), **getopt**(C), **wait**(S)

adb

invokes a general-purpose debugger.

Syntax

adb [-w] [-p *prompt*] [*objfil* [*corefile*]

Description

adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of binaries.

objfil is normally an executable file of either XENIX or COFF format, preferably containing a symbol table; if not then the symbolic features of **adb** cannot be used although the file can still be examined. The default filename for *objfil* is *a.out*. *corefile* is assumed to be a core image file produced after executing *objfil*; the default for *corefile* is *core*.

Requests to **adb** are read from standard input. If the **-w** option is specified, both *objfil* and *corefile* are created if necessary and opened for reading and writing. The QUIT and INTERRUPT keys cause **adb** to return to the next command. The **-p** option defines the *prompt* string, which may be any combination of characters. The default is an asterisk (*).

In general requests to **adb** are of the form:

[*address*] [, *count*] [*command*] [;]

If *address* is present then the current position in the file (called *dot*) is set to the value given by *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *address* is a special expression having the form:

[*segment*:]*offset*

where *segment* gives the address of a specific text or data segment, and *offset* gives an offset from the beginning of that segment. If *segment* is not given, the last segment value given in a command is used.

The interpretation of an address depends on the context in which it is used. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see "Addresses".

Expressions

. The value of ***dot***.

+ The value of ***dot*** incremented by the current increment.

^ The value of ***dot*** decremented by the current increment.

" The last ***address*** typed.

integer An octal number if ***integer*** begins with a 0; a hexadecimal number if preceded by # or 0x; otherwise a decimal number.

integer.integer
A 32-bit floating point number.

´cccc´ The ASCII value of up to 4 characters. "\" may be used to escape a ´.

<name The value of ***name***, which is either a variable name or a register name. **adb** maintains a number of variables (see "Variables") named by single letters or digits. If ***name*** is a register name then the value of the register is obtained from the system header in ***corefile***. The 286 register names are **ax bx cx dx di si bp fl ip cs ds ss es sp**. The 386 register names are **eax ebx ecx edx edi esi ebp esp**. The name **fl** refers to the status flags.

symbol A ***symbol*** is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. "\" may be used to escape other characters. The value of the ***symbol*** is taken from the symbol table in ***objfil***. An initial _ or ~ will be prepended to ***symbol*** if needed.

_symbol
In C, the "true name" of an external symbol begins with an underscore (_). It may be necessary to use this name to distinguish it from internal or hidden variables of a program.

(*exp*) The value of the expression ***exp***.

Monadic operators

exp* The contents of the location addressed by ***exp*.

-exp Integer negation.

~exp Bitwise complement.

Dyadic operators

Dyadic operators are left-associative and are less binding than monadic operators.

e1+e2 Integer addition.

e1-e2 Integer subtraction.

*e1*e2* Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1|e2 Bitwise disjunction.

e1^e2 Remainder after division of *e1* by *e2*.

e1#e2 *e1* rounded up to the next multiple of *e2*.

Commands

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands ? and / may be followed by *; see "Addresses" for further details.)

?f Locations starting at text *address* in *objfil* are printed according to the format f.

/f Locations starting at data *address* in *corefile* are printed according to the format f.

=f The value of *address* itself is printed in the styles indicated by the format *f*. (For i format "?" is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented temporarily by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows:

Letter	Format
o	2 bytes in octal
O	4 bytes in octal
q	2 bytes in signed octal
Q	4 bytes in signed long octal
d	2 bytes in decimal
D	4 bytes in decimal
x	2 bytes in hexadecimal
X	4 bytes in hexadecimal
u	2 bytes as an unsigned integer
U	4 bytes as an unsigned integer
f	4 bytes in floating point
F	8 bytes in floating point
b	1 address byte in octal
c	1 byte as a character
C	1 byte as a character using the following escape convention. Character values 000 to 040 are printed as an at-sign (@) followed by the corresponding character in octal range 0100 to 0140. The at-sign character itself is printed as @@.
s	Prints the addressed characters until a zero character is reached.
S	Prints a string using the at-sign (@) escape convention When *n* is a length of the string including its zero terminator.
Y	4 bytes in date format. See ctime(S).
i	Machine instructions. n is the number of bytes occupied by the instruction. This style of printing causes variable 1 and 2 to be set the the offset parts of the source and destination respectively.
a	The current symbolic address. Symbols are checked to ensure that they have an appropriate type as indicated below: / local or global data symbols ? local or global text symbols = local or global absolute symbols
A	The current absolute address
p	Prints the addressed value in symbolic form using the same rules for symbol lookup as a
t	A horizontal TAB
r	A blank space
n	A Newline
'...'	Prints the enclosed string
^	Decrements *dot* by the current increment. Nothing is printed
=	Increments *dot* by 1. Nothing is printed
-	Decrements *dot* by 1. Nothing is printed
+	Increments *dot* by 1. Nothing is printed

newline
If the previous command temporarily incremented *dot*, makes the increment permanent. Repeat the previous command with a *count* of 1.

[?/]l *value* **mask**
Words starting at *dot* are masked with **mask** and compared with *value* until a match is found. If **L** is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is set to the matched location. If **mask** is omitted then -1 is used.

[?/]w *value* ...
Writes the 2-byte *value* into the addressed location. If the command is **W**, writes 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m *segnum fpos size*
Sets new values for the given segment's file position and size. If *size* is not given, then only the file position is changed. The *segnum* must the segment number of a segment already in the memory map (see "Addresses"). If **?** is given, a text segment is affected; if **/** a data segment.

[?/]M *segnum fpos size*
Creates a new segment in the memory map. The segment is given file position *fpos* and physical size *size*. The *segnum* must not already exist in the memory map. If **?** is given, a text segment is created; if **/** a data segment.

>*name*
dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following !.

$*modifier*
Miscellaneous commands. The available *modifiers* are:

- **<f** Read commands from the file **f** and return.
- **>f** Send output to the file **f**, which is created if it does not exist.
- **r** Print the general registers and the instruction addressed by *ip*. *Dot* is set to *ip*.
- **f** Print the floating registers in single or double length.
- **b** Print all breakpoints and their associated counts and commands.
- **c** C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of *bp*). If *count* is given then only the first *count* frames are printed. This command is case insensitive.
- **e** The names and values of external variables are printed.
- **w** Set the page width for output to *address* (default 80).
- **s** Set the limit for symbol matches to *address* (default 255).
- **o** Sets input and output default format to octal.

d Sets input and output default format to decimal.

x Sets input and output default format to hexadecimal.

q Exit from **adb**.

v Print all non zero variables in octal.

m Print the address map.

:*modifier*

Manage a subprocess. Available modifiers are:

br Set breakpoint at *address*. The breakpoint is executed *count*-1 times before causing a stop. Each time the breakpoint is encountered the command **c** is executed. If this command sets *dot* to zero then the breakpoint causes a stop.

dl Delete breakpoint at *address*.

r[*arguments*]

Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. *arguments* to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.

R[*arguments*]

Same as the **r** command except that *arguments* are passed through a shell before being passed to to the program. This means shell metacharacters can be used in filenames.

co The subprocess is continued and signal **s** is passed to it, see **signal**(S). If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.

s As for **co** except that the subprocess is single stepped **count** times. If there is no current subprocess then *objfil* is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

k The current subprocess, if any, is terminated.

Variables

adb provides a number of variables. Named variables are set initially by **adb** but are not used subsequently. Numbered variables are reserved for communication as follows.

0 The last value printed.
1 The last offset part of an instruction source.
2 The previous value of variable 1.

On entry the following are set from the system header in the *corefile*. If *corefile* does not appear to be a *core* file then these values are set from *objfil*:

b The base address of the data segment.
d The data segment size.
e The entry point.
m The execution type.
n The number of segments.
s The stack segment size.
t The text segment size.

Addresses

Addresses in **adb** refer to either a location in a file or in actual memory. When there is no current process in memory, **adb** addresses are computed as file locations, and requested text and data are read from the *objfil* and *corefile* files. When there is a process, such as after a **:r** command, addresses are computed as actual memory locations.

All text and data segments in a program have associated memory map entries. Each entry has a unique *segment number*. In addition, each entry has the *file position* of that segment's first byte, and the *physical size* of the segment in the file. When a process is running, a segment's entry has a *virtual size* which defines the size of the segment in memory at the current time. This size can change during execution.

When a address is given and no process is running, the file location corresponding to the address is calculated as:

effective-file-address = file-position + offset

If a process is running, the memory location is simply the offset in the given segment. These addresses are valid if and only if

0 <= offset <= size

where *size* is physical size for file locations and virtual size for memory locations. Otherwise, the requested *address* is not legal.

The initial setting of both mappings is suitable for normal *a.out* and *core* files. If either file is not of the kind expected then, for that file, ***file position*** is set to 0, and ***size*** is set to the maximum file size. In this way, the whole file can be examined with no address translation.

So that **adb** may be used on large files, all appropriate values are kept as signed 32 bit integers.

Files

a.out
core

Diagnostics

The message "adb" appears when there is no current command or format.

Comments about inaccessible files, syntax errors, abnormal termination of commands, etc.

Exit status is 0, unless last command failed or returned nonzero status.

Notes

A breakpoint set at the entry point is not effective on initial entry to the program.

System calls cannot be single stepped.

Local variables whose names are the same as an external variable may foul up the accessing of the external.

COFF or *x.out* format files are accepted and read transparently.

See also

ptrace(S), *a.out*(FP), *core*(FP)

admin

create and administer SCCS files

Syntax

admin [[-n] [[-i[*name*]] [-r*rel*]] [-t*name*] [-m[*mrlist*]] [-y[*comment*]]] *files*
admin [-t[*name*]] [-f*flag*[*value*]] [-a*login*] [-e*login*] *files*

admin [-t[*name*]] [-f*flag*[*value*]] [-d*flag*[*value*]] [-a*login*] [-e*login*] [-h] [-z] *files*

Description

The **admin** command is used to create new SCCS files or to change parameters of existing ones. Arguments to **admin**, which may appear in any order, consist of keyletter arguments and named files. (Note that SCCS file names must begin with the characters "*s.*".)

If a named file does not exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as they are.

If a directory is named, **admin** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of "-" is given, standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The options and arguments follow. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

-n Creates a new SCCS file.

-i[*name*] Creates a new SCCS file, taking the text from the file *name* or from standard input if the filename is omitted. The text constitutes the first delta of the file (see the **-r** option for delta numbering scheme). If this option is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an **admin** command on which the **-i** option is supplied. Using a single **admin** command to create two or more SCCS files requires that they be created empty (no **-i** option). Note that **-i** implies **-n**.

-r*rel* The release into which the initial delta is inserted; if not specified it is inserted into release 1. This option may be used only if **-i** is also used. The level of the initial delta is always 1. (Initial deltas are named 1.1 by default.)

-t[*name*] Inserts, removes, or replaces descriptive text in an SCCS file. The *name* argument specifies a file from which the descriptive text is to be taken. When creating a new SCCS file (using **-n** and/or **-i** options), the *name* must be provided. Remove any descriptive text currently in an existing SCCS file, by specifying **-t** without an argument. Replace any descriptive text currently in an existing SCCS file, by specifying a file *name* which contains the replacement text.

-f*flag* This option specifies a *flag*, and possibly a value for the *flag*, to be placed in the SCCS file. The **-f** option may be supplied several times. The allowable *flag*s and their values are:

b Allows use of **get -b** to create branch deltas.

cceil The highest release (the "ceiling"), a number greater than 0 but less than or equal to 9999, which may be retrieved by a **get**(CP) command for editing. The default value for an unspecified **c** flag is 9999.

ffloor The lowest release (the "floor"), a number greater than 0 but less than 9999, which may be retrieved by a **get**(CP) command for editing. The default value for an unspecified **f** flag is 1.

d*SID* The default delta number (*SID*.1) to be used by a **get**(CP) command.

i[*str*] Causes the `No id keywords (ge6)` message issued by **get**(CP) or **delta**(CP) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string; however, the string must contain a keyword and no embedded newlines.

j Allows concurrent **get**(CP) commands for editing on the same *SID* of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l*list* A *list* of releases to which deltas can no longer be made (**get -e** against one of these "locked" releases fails). The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range>::=     | a
```

The character " a " in the *list* is equivalent to specifying all releases for the named SCCS file.

n Causes **delta**(CP) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a new release (for example, in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be nonexistent in the SCCS file, preventing branch deltas from being created from them in the future.

q*text* User-definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by **get**(CP).

m*mod* Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by **get**(CP). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading *s.* removed.

t*type* *Type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by **get**(CP).

v*pgm* Causes **delta**(CP) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program (see **delta**(CP)). (If this flag is set when creating an SCCS file, the **-m** option must also be used even if its value is null.)

-d*flag* Causes removal (deletion) of the specified *flag* from an SCCS file. The **-d** option may be specified only when processing existing SCCS files. The **-d** option may be specified several times on a single **admin** command. See the **-f** keyletter for allowable *flag* names.

l*list* A *list* of releases to be "unlocked". See the **-f** keyletter for a description of the l flag and the syntax of a *list*.

-a*login* A *login* name or numerical system group ID to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** keyletters may be used on a single **admin** command line. As many *logins* or numerical group IDs as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If *login* or group ID is preceded by a "!" it is to be denied permission to make deltas.

-e*login* A *login* name or numerical group ID to be erased from the list of users allowed to make deltas (changes) to the SCCS file.

Specifying a group ID is equivalent to specifying all ***login*** names common to that group ID. Several **e** keyletters may be used on a single **admin** command line.

-m[*mrlist*] The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to **delta**(CP). The **v** flag must be set; the MR numbers are validated if the **v** flag has a value (the name of an MR number validation program). Diagnostics will occur if the **v** flag is not set or MR validation fails.

-y[*comment*] The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of **delta**(CP). Omission of the **-y** keyletter results in a default comment line being inserted in the form:

date and time created YY/MM/DD HH:MM:SS by login

The **-y** keyletter is valid only if the **-i** and/or **-n** keyletters are specified (that is, a new SCCS file is being created).

-h Causes **admin** to check the structure of the SCCS file and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-z The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

Note that use of this keyletter on a corrupted file may prevent future detection of the corruption.

The last component of all SCCS file names must be of the form *s.filename*. New SCCS files are created in read-only mode. Write permission in the directory is required to create a file. All writing done by **admin** is to a temporary x-file, called *x.filename* (see **get**(CP)) created in read-only mode if creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of **admin**, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

We recommend that directories containing SCCS files be writable only by the owner, and that SCCS files themselves be read-only. This will ensure that only the owner can modify the files, and prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed by the owner. *Care must be taken!* The edited file should *always* be processed by an **admin -h** to check for corruption followed by an **admin -z** to generate a proper check-sum. Another **admin -h** is recommended to ensure the SCCS file is valid.

The **admin** command also makes use of a transient lock file (called *z.filename*), which is used to prevent simultaneous updates to the SCCS file by different users. See **get**(CP) for further information.

Files

d-file	Created during the execution of **delta**; removed after completion of **delta**.
g-file	Existed before the execution of **delta**; removed after completion of **delta**.
p-file	Existed before the execution of **delta**; may exist after completion of **delta**.
q-file	Created during the execution of **delta**; removed after completion of **delta**.
x-file	Created during the execution of **delta**; renamed to SCCS file after completion of **delta**.
z-file	Created during the execution of **delta**; removed during the execution of **delta**.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

See also

delta(CP), **ed**(C), **get**(CP), **prs**(CP), **sccsfile**(FP), **what**(CP)

Standards conformance

admin is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

ar

archive and library maintainer for portable archives

Syntax

ar *key* [*keyarg*] [*posname*] *afile* [*name*] ...

Description

The **ar** command combines groups of files in a single archive file. Its primary use is to create and update linker library files, but it can be used for any similar purpose. **ar**'s magic string and the file headers consist of printable ASCII characters, so if an archive is composed of printable files, the entire archive can be printed. Archives of text files created by **ar** are portable between implementations of System V.

The portable archive format and structure is described in detail in the *ar*(FP) manual page. The archive symbol table is used by **ld** to effect multiple passes over libraries of object files.

An archive symbol table is only created and maintained by **ar** when there is at least one object file in the archive. The archive symbol table is in a specially named file that is always the first file in the archive (this file is not accessible to the user). Whenever **ar** is used to create or update the contents of an archive, the symbol table is rebuilt.

The long filename map is the second file in the archive (this file only exists if one or more archived files have long filenames). Previously, the **ar** format permitted archive members to have file names of up to 14 characters in length. Now longer file names are allowed. Thus, *ar*(FP) was changed to permit archiving files with such long names. In order to accommodate the longer file names without breaking backwards compatibility, another specially-named archive member, the long filename map, was introduced.

The long filename map comes right after the symbol table if any of the archived files have names longer than 14 characters. This means that both the symbol table and the longfilename map are optional, but when they both appear the longfilename map appears after the symbol table. The longfilename map contains a one-to-one mapping between the real filenames and coded files names that are used internally by **ar**. These coded names are not accessible by the user.

Unlike command options, the *key* is a required part of **ar**'s command line. The *key* (which may begin with a "-") is formed with one of the following letters: **dmpqrtx**. Arguments to the *key* are made with one or more of the following: **abcilsuv**. *posname* is an archive member name used as a reference point in positioning other files in *afile*, the archive file. The *names* are constituent files in the archive file.

The valid ***key*** characters are:

⟨d⟩ Delete the named files from the archive file.

⟨m⟩ Move the named files to the end of the archive. If a positioning character is present, then the ***posname*** argument must be present and, as in ⟨r⟩, specify where the files are to be moved.

⟨p⟩ Print the named files in the archive.

⟨q⟩ Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. This option is useful to avoid quadratic behavior when creating a large archive piece-by-piece. Unchecked, the file may grow exponentially up to the second degree.

⟨r⟩ Replace the named files in the archive file. If the optional character ⟨u⟩ is used with ⟨r⟩, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set *abi* is used, then the ***posname*** argument must be present and specify that new files are to be placed after (⟨a⟩) or before (⟨b⟩ or ⟨i⟩) ***posname***. Otherwise new files are placed at the end.

⟨t⟩ Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

⟨x⟩ Extract the named files. If no names are given, all files in the archive are extracted. In neither case does ⟨x⟩ alter the archive file.

The meanings of the ***key*** arguments are as follows:

⟨a⟩ Used with the ⟨m⟩ or ⟨r⟩ ***key*** characters, it specifies that new files are to be placed after ***posname*** in the archive file.

⟨b⟩ Used with the ⟨m⟩ or ⟨r⟩ ***key*** characters, it specifies that new files are to be placed before ***posname*** in the archive file.

⟨c⟩ Suppress the message that is produced by default when ***afile*** is created.

⟨i⟩ Used with the ⟨m⟩ or ⟨r⟩ ***key*** characters, it specifies that new files are to be placed before ***posname*** in the archive file.

⟨l⟩ Place temporary files in the local (current working) directory rather than in the default temporary directory, *TMPDIR*.

⟨s⟩ Force the regeneration of the archive symbol table even if **ar**(CP) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the **strip**(CP) command has been used on the archive.

⟨u⟩ When used with the ⟨r⟩ ***key*** character, then only updated versions of files in the archive are used to overwrite the originals.

⟨v⟩ Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with ⟨t⟩, give a long listing of all information about the files. When used with ⟨x⟩, precede each file with a name.

Files

$TMPDIR/* temporary files

$TMPDIR is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**( ) in **tmpnam**(S)).

Notes

If you are using XENIX binaries, please refer to the **ar**(XNX) manual page for information on use with XENIX binaries.

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

This utility determines whether an object file is in OMF or COFF format and executes the appropriate binaries in each case. Since this utility relies on the files in */lib/coff* and */lib/xout*, changing or moving any of those files results in an error.

See also

a.out(FP), **ar**(FP), **ld**(CP), **lorder**(CP), **strip**(CP), **tsort**(CP), **tmpnam**(S)

Standards conformance

ar is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

as

common assembler

Syntax

as [-dl] [-m] [-n] [-o *objfile*] [-R] [-V] [-Y [*md*],*dir*] *filename*

Description

The **as** command assembles the named file. The following flags may be specified in any order:

-dl Do not produce line number information in the object file.

-m Run the **m4** macro processor on the input to the assembler.

-n Turn off long/short address optimization. By default, address optimization takes place.

-o *objfile*
Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the *.s* suffix, if there is one, from the input file name and appending a *.o* suffix.

-R Remove (unlink) the input file after assembly is completed.

-V Write the version number of the assembler being run on the standard error output.

-Y [*md*],*dir*
Find the **m4** preprocessor (*m*) and/or the file of predefined macros (*d*) in directory *dir* instead of in the customary place. There must be a space between the **-Y** flag and its options.

Files

TMPDIR/* temporary files

TMPDIR is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**( ) in **tmpnam**(S)).

Warning

If the **-m** (**m4** macro processor invocation) option is used, keywords for **m4** cannot be used as symbols (variables, functions, labels) in the input file since **m4** cannot determine which are assembler symbols and which are real **m4** macros.

Notes

The assembler directive may not work in the **.text** section when optimization is performed.

Arithmetic expressions may only have one forward referenced symbol per expression.

Wherever possible, the assembler should be accessed through a compilation system interface program such as **cc**(CP) or **rcc**(CP).

See also

a.out(FP), **cc**(CP), **ld**(CP), **m4**(CP), **nm**(CP), **strip**(CP), **tmpnam**(S)

Standards conformance

as is conformant with:

AT&T SVID Issue 2.

cb

C program beautifier

Syntax

cb [-s] [-j] [-l *leng*] [-t *leng*] [*file* ...]

Description

The **cb** command reads C programs either from *file* or from standard input and writes them on standard output with spacing and indentation that display the structure of the code. Under default options, **cb** preserves all user newlines.

cb accepts the following options:

-s Formats the code in the style of Kernighan and Ritchie in *"The C Programming Language."*

-j Rejoin split lines.

-l *leng* Split lines that are longer than *leng*. By default, **cb** splits lines that are longer than 120 characters.

-t *leng* Interpret tabs as being *leng* characters long. By default, **cb** treats tabs as being 8 characters in length.

Notes

Punctuation that is hidden in preprocessor statements will cause indentation errors.

See also

cc(CP)

Kernighan, B. W., and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, 1978.

cc

invokes the C compiler.

Syntax

cc [*option*]...*file*... [*option*...*file*...] [**-link**[*link-libinfo*]]

Description

cc is the C compiler command, and is based on Microsoft's C compiler version 6. It creates executable programs by compiling and linking the files named by the *file* arguments, and copies the resulting program to *a.out* (provided that the **-c** or **-o** options are not used), which is in Common Object File Format (COFF) by default. To specify the generation of XENIX Object Module Format (OMF) files, use the **-xenix**, **-x2.3**, **-xout**, **-M0**, **-M1**, **-M2**, or **-compat** options.

You can direct the compiler to produce DOS or OS/2 binaries by using the **-dos** or **-os2** options. If one of these is present, it overrides all other options on the command line. If both are present, then the one that appears first overrides the other. Therefore, if you place both **-xenix** and **-dos** on your command line, the output will be in DOS format.

The *file* can be any C or assembly language source file or any object or library file. C source files must have a *.c* extension. Assembly language source files must have *.s* or *.asm*, object files *.o*, and library files *.a* extensions. **cc** invokes the C compiler for each C source file and copies the result to an object file whose base name is the same as the source file but whose extension is *.o*. The **cc** command invokes the AT&T assembler, **as**, for each assembly source file that ends with *.s*, and it invokes the Microsoft assembler, **masm**, for each assembly source file that ends with *.asm*. The **cc** command copies the result to an object file with extension *.o*. **cc** ignores object and library files until all source files have been compiled or assembled. It then invokes the link editor, **ld**, (provided the **-c** option is not used) and combines all the object files it has created together with object files and libraries given in the command line to form a single program.

Files are compiled as they occur on the command line, and only options that are listed before the name of the file on the command line affect the processing of that file. This makes the order of files very important. The exceptions to this are options to the linker, such as **-l**. These are taken into account during linking after all object files to be used in the link are gathered. A library file specified on a **-l** option may be used to resolve an undefined symbol reference only if the **-l** appears later than the file containing the unresolved symbol.

A number of standard libraries are searched automatically. These libraries support the standard C library functions and program startup routines. Which libraries are used depends on the program's specific options and memory model. By default, the libraries are found in the directories */lib* and */usr/lib* and their names are prefixed with *"lib"*. The entry point of the resulting program is set to the beginning of the standard startup code which then calls the **main( )** function of the program.

By default, **cc** creates binaries with full international functionality; the library file, *libintl.a*, is automatically included in the link command line, and **M_INTERNAT** and **_M_INTERNAT** are automatically defined by **cc**. The **-nointl** option cancels international functionality.

NOTE A number of options listed below cause a file to be created, the name of which consists of the basename of the source file being processed plus an extension. (The basename is everything up to the last occurrence of the dot character (.) in the filename.) In the discussion below, these files will be referred to as "source.*ext*", where *ext* is the extension.

These are the available options:

-ansi
Enforces full ANSI conformance. The language is restricted to ANSI specifications. The ANSI conformant library is used in the link phase. When using this option, you don't need to link in the math library (with **-lm**), because the math routines are already included in the conformant library and are automatically linked in. If you specify both **-ansi** and **-lm**, you will be warned about undefined symbols. Use the **-strict** or **-Za** flags instead if you want to link with non-conforming libraries. Do not use the **-posix** or **-xpg3** flags with this flag.

-B1 *path/filename*
Define an alternate pass 1 (p1) for the compiler.

-B2 *path/filename*
Define an alternate pass 2 (p2) for the compiler.

-B3 *path/filename*
Define an alternate pass 3 (p3) for the compiler.

-Ba *path/filename*
Define an alternate assembler.

-Bl *path/filename*
Define an alternate loader.

-Bm *path/filename*
Define an alternate pass **m4** preprocessor when profiling code.

-c Creates a linkable object file for each source file but does not call the linker. No executable program is created.

-compat
Makes an executable file that is binary compatible across the following systems (as distributed by certain vendors):

386 UNIX System V Release 3.2
UNIX-286 System V
UNIX-386 System V
UNIX-286 3.0
UNIX-8086 System V

This causes XENIX libraries to be used, and creates OMF object files.

-C Preserves comments when preprocessing a file with **-E**, **-P**, or **-EP**. That is, comments are not removed from the preprocessed source. This option may only be used in conjunction with **-E**, **-P**, or **-EP**.

-CSON, -CSOFF
When optimization (**-O**) is specified, these options enable (**-CSON**) or disable (**-CSOFF**) "common sub-expression" optimization.

-d Displays the compiler passes and their arguments before they are executed.

-dos Directs **cc** to create an executable program for MS-DOS systems. The DOS libraries are used with this option. If this is present, it overrides all other options on the command line. The only exception to this is **-os2**, which will override **-dos** if it appears earlier on the command line.

-D*name*[=*string*]
Defines *name* to the preprocessor as if defined by a **#define** statement in each source file. The format "**-D***name*" sets *name* to a value of 1. The format "**-D***name=string*" sets *name* to the given *string*. See the **-U** option for more information on the default manifest defines. The **-D** option can be used several times on the command line, but it must not define the same *name* twice.

-E Preprocess each source file as described for **-P**, but copy the result to standard output. It also places a **#line** directive with the current input line number and source file name at the beginning of output for each file.

-EP Preprocess each source file as described for **-P**, and print the output both to a file and standard output. **-EP** does not place a **#line** directive at the beginning of the file.

-F*num*
Sets the size of the program stack to *num* bytes. The value of *num* must be specified in hexadecimal. The default stack for the 8086 is variable, starting at the top of a full 64 Kbyte data segment that grows down until it reaches data. The default stack for the 80286 is 4096 bytes (0x1000 hexadecimal). This option does not apply to the 80386, which has a variable stack.

-Fa, -Fa*filename*
Create an assembly source listing in source.*asm* or *filename* in MASM format. Continues with the link if requested. This performs the same function as the **-S** option.

-Fc, -Fc*filename*
Create a merged assembler and C listing in source.*L* or in *filename*.

-Fe*filename*
Names the executable program file *filename*.

-Fl, -Fl*filename*
Create a listing file in source.*L* (or *filename*) with assembly source and object code. Continues with the link if requested.

-Fm, -Fm*filename*
Instruct the linker to direct a map listing to *a.map* (or to *filename*). This file contains the names of all segments in order of their appearance in the load module. Use with the **-xenix** option to produce XENIX-compatible binaries.

-Fo*object-filename*
The **-Fo** option, together with the *object-filename*, must precede a source filename; it causes the object file created from that source to be called *object-filename* instead of the default name, source.*o*. There may be multiple **-Fo** options on the command line, each acting on the source file that it precedes.

-FPa, -FPc, -FPc87, -FPi, -FPi87
When used in conjunction with **-dos** or **-os2**, these options control the type of floating point code generated and which library support to use. The default is **-FPi**.

-Fs, -Fs*filename*
Creates a C source listing in source.*S* or in *filename*.

-g Includes information for the screen-oriented debuggers **dbxtra** and **CodeView** and the symbolic debugger (**sdb**) in the output file. (This is equivalent to using the **-Zi** option.)

-Gc Specifies the alternative calling sequence and naming conventions used in System V 386 Pascal and System V 386 FORTRAN. For more information, see the *Programming Tools Guide*.

-Gs Removes stack probe routines to reduce binary size and speed execution slightly. Stack probes are used to detect stack overflow on entry to program routines. Use of this option disables stack checking and stack overflow can occur without warning or error messages. Code generated for the 80386 processor does not require stack probes, therefore this option has no effect if **-M3** is specified. **-Gs** is equivalent to the **-K** option.

-H*len*
Sets the maximum length of external symbols to *len*. Names longer than *len* are truncated before being copied to the external symbol table. The maximum length, that *len* can be set to, is 254 characters. This is equivalent to the -nl option.

-help, -HELPC
Prints help menu.

-i Creates separate instruction and data spaces for small model programs (8086/186/286 compilations only). When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and may be shared by all users executing the file. This option is implied when creating medium, large, or huge model programs.

-I *pathname*
Adds *pathname* to the list of directories to be searched when an **#include** file is not found in the directory containing the current source file or whenever angle brackets (< >) enclose the filename. If the file cannot be found in directories in this list, directories in a standard list are searched. **-I** can be used several times, but only applies to source files appearing after it on the command line.

-iBCS2
Enforces strict Intel Binary Compatibility Standard 2 conformance. This flag automatically turns on the **-Of** flag.

-J Changes the default mode for the *char* type from signed to unsigned.

-K Remove stack probes from a program, to reduce binary size and speed execution slightly. Stack probes are used to detect stack overflow on entry to program routines. Use of this option disables stack checking and stack overflow can occur without warning or error messages. Code generated for the 80386 processor does not require stack probes, therefore this option has no effect if **-M3** is specified. **-K** is equivalent to the **-Gs** option.

-l*name*
Searches library *libname.a* for unresolved function references.

-link All options and filenames that follow **-link** (and any options not recognized by the compiler) are passed directly to **ld**. This option must appear last on the **cc** command line.

-L Creates an assembler listing containing assembled code and assembly source instructions in the file source.*L*. **-L** suppresses the **-S** and **-Fa** options. In order to add directories to the path that the linker searches, use **-link -L**.

-LARGE
Invokes the large model passes of the compiler. This option is included in this release only for backward compatibility.

-m *name*
Creates a map file called *name*. Use this option only to produce XENIX-compatible binaries. This is equivalent to the **-Fm** option.

-M *string*
Sets the program configuration. This configuration defines the program's memory model, word order, and data threshold. It also enables C language enhancements such as advanced instruction set and keywords. The number specifying the CPU type (0, 1, 2, or 3) must be given before the letter specifying the size model of the program. For example, to compile a large model 286 program, the option should be: **-M2l**.

The *string* may be any combination of the following (**s**, **m**, **l**, and **h** are mutually exclusive):

0 8086 code generation. OMF object files are created. The **s**, **m**, and **l** specifiers are valid here.

1 186 code generation. OMF object files are created. The **s**, **m**, **l**, and **h** specifiers are valid here.

2 286 code generation for compiled C source files. OMF object files are created. The **s**, **m**, **l**, and **h** specifiers are valid here.

3 Enables 386 code generation (the default). Either COFF or OMF object files can be created. The **m**, **l**, and **h** specifiers are not valid here: only the small model can be used for 386 programs.

b Reverses the word order for *long* types, putting the high order word first; the default is low order word first.

c Creates a compact model program.
For 186/286 compilations only.

d Instructs the compiler not to assume register SS=DS.

e Enables the **far**, **near**, **huge**, **pascal**, and **fortran** keywords. Also enables certain non-ANSI extensions necessary to ensure compatibility with existing versions of the C compiler.

f Enables software floating point that does not exist in UNIX. Useful when compiling object files to be linked on MS-DOS.

h Creates a huge model program.
For 186/286 compilations only.

l Creates a large model program.
For 8086/186/286 compilations only.

m Creates a medium model program.
For 8086/186/286 compilations only.

s Creates a small model program (the default model).

t *num* Sets the threshold for the size of the largest data item in the data group to *num*. The default is 32,767. This option can only be used in large model programs.

Specifying CPU type "0", "1", or "2" produces OMF files and uses the OMF libraries appropriate to the model size. See the **-xenix** option for more information.

-n Sets pure text model. Pure text is the only model supported for 80386 binaries; this is achieved without separate instruction and data space. For 8086, 80186, or 80286 binaries this option is equivalent to **-i** and will give a warning that it is setting **-i** when used.

-nl *len*
Sets the maximum length of external symbols to *len*. Names longer than *len* are truncated before being copied to the external symbol table. The maximum length, that *len* can be set to, is 254 characters. (This is equivalent to using the **-H** option.)

-nointl
Directs **cc** to create a binary that does not include international functionality.

-norespawn
Prevents the compiler from restarting if any of the **-O** optimizations fail. By default, the compiler respawns itself on a failure with the optimization that caused the failure turned off.

-ND *name*
Sets the data segment name for each compiled or assembled source file to *name*. If **-ND** is not given, the name "_DATA" is used.

In large model programs (**-Ml**) the **-ND** option can only be used on "leaf modules" - those that make no calls to routines in another segment.

-NM *name*
Sets the module name for each compiled or assembled source file to *name*. If not given, the filename of each source file is used.

-NT *name*
Sets the text segment name for each compiled or assembled source file to *name*. If not given, the name "*module*_TEXT" is used for medium model and "_TEXT" for small model programs. This option should not be used on 386 code.

-o *filename*
Defines *filename* to be the name of the final executable program. This overrides the default name *a.out*.

-os2 Directs **cc** to create an executable program for OS/2, using OS/2 libraries. If **-os2** is present, it overrides all other options on the command line. The only exception to this is **-dos**, which will override **-os2** if it appears earlier on the command line.

-O *string*

Invokes the object code optimizer. The ***string*** consists of one or more of the following characters:

3 Disable pass 3 (p3) optimization.

a Relax alias checking.

c Enable default (block-level) local common expressions.

d Turns off the following optimizations (if they are on): **a,c,e,g,i,l,s,t**. It also turns on the **p** flag to improve consistency in floating-point calculations

e Enable global register allocation.

f Causes the compiler to treat single precision floating-point (float) arguments as single precision if a function prototype declares the parameter to be float. It also causes the compiler to treat a float parameter declared within a parameter type list as if it were a float within the called function. Note: it is essential that the file with the function call and the file with the function body both be compiled identically with respect to this option. Older releases of the compiler may treat single precision floating-point arguments differently; for more details, see the Development System *Release and Installation Notes.*

g Enable global optimizations and global common expressions.

h Optimizes code for functions returning type ***char*** or ***short***. If this flag is not used the types will be widened to ***int*** (386 only).

i Enable generation of intrinsic routines.

l Perform loop optimizations (386 only).

m Causes the compiler (386/486 only) to optimize the calling sequence for a function returning a small structure (1, 2, or 4 bytes in size). Note: it is essential that the file with the function call and the file with the function body both be compiled identically with respect to this option. Older releases of the compiler may optimize the calling sequence differently; for more details, see the Development System *Release and Installation Notes.*

n Disable unsafe loop optimizations (default).

p Improve consistency in floating-point calculations.

r Disable in-line returns from functions.

s Optimize code for space.

t Optimize code for speed (default).

x Perform maximum optimization. To determine which optimizations are being performed, use the **-z** option to **cc**.

z Enable maximum loop and global-register allocation optimization.

If the **-O** flag is not specified at all then the **t** and **c** optimizations are turned on by default.

-p Adds code for program profiling to the executable program. Profiling code counts the number of calls to each routine in the program; specific lines in programs can be profiled by marking them using the MARK( ) macro. When you execute the program, the profiling information is copied to the file *mon.out*. This file can be examined using **prof**(CP).

-P Preprocesses each source file and copies the result to a file called source.*i* (the same base name as the source).

-pack
Packs structures. Each structure member is stored at the first available byte, without regard to ***int*** boundaries. Although this will save space, execution will be slower because of the extra time required to access 16-bit members that begin on odd boundaries.

-posix
Enforces strict POSIX conformance. The language is restricted to ANSI specifications. The POSIX and ANSI conformant C libraries are used in the link phase. When using this option, you don't need to link in the math library (with **-lm**), because the math routines are included in the conformant libraries and are automatically linked in. If you specify both **-posix** and **-lm**, you will be warned about undefined symbols. Do not use the **-ansi** or **-xpg3** flags with this flag.

-ql Provides line profiling, that is, "coverage" and "count" statistics for each line of the program. The output from line profiling can be examined with **lprof**(CP).

-qp This flag is equivalent to **-p**.

-quiet
Turns off the echoing of source filenames to standard error during compilation. The environment variable **QUIETCC** can be used to suppress the source filename echo for single-file compilations. For multiple-file compilations, **QUIETCC** has no effect: source filenames are echoed normally. **QUIETCC** overrides the **-quiet** option.

-r Performs an incremental link.

-s Instructs the linker to strip symbol table information from the executable output file.

-S Creates an assembly source listing in source.*asm*. It should be noted that this file is in **masm** format. This option performs the same function as **-Fa**.

-Sl *linewidth*
Sets the maximum number of characters per line in the source listing (the default value is 79 characters per line). This option only takes effect when **-Fs** is used.

-Sp *pagelength*
Sets the number of lines per page in the source listing. This option only takes effect when **-Fs** is used.

-Ss *string*, **-St** *string*
Sets subtitle (**-Ss**) and title (**-St**) for source listings and bypasses **cc**'s linking operation. These options only take effect when the **-Fs** option is also used to create a source listing.

-strict
Restricts the language to ansi specifications. The standard C library is used in the link phase. Use the **-ansi** flag instead if the ANSI conformant C library is required in the link phase.

-svid
Enforces SVID Issue 2 conformance.

-Tc *filename*
Tells **cc** that the given *filename* is a C source file. Use this option if the C source file does not have the *.c* extension.

-u Eliminates all manifest defines.

-U*definition*

Removes or undefines the given manifest define. Manifest defines are predefined macros that the user should not redefine. The manifest defines are as follows:

i386	__i386
M_BITFIELDS	_M_BITFIELDS
M_COFF	_M_COFF
M_I386	_M_I386
M_I8086	_M_I8086
M_I186	_M_I186
M_I286	_M_I286
M_I86	_M_I86
M_I86CM	_M_I86CM
M_I86HM	_M_I86HM
M_I86LM	_M_I86LM
M_I86MM	_M_I86MM
M_I86SM	_M_I86SM
M_INTERNAT	_M_INTERNAT
M_LDATA	_M_LDATA
M_LTEXT	_M_LTEXT
M_SDATA	_M_SDATA
MSDOS	
M_STEXT	_M_STEXT
M_SYS3	_M_SYS3
M_SYS5	_M_SYS5
M_SYSIII	_M_SYSIII
M_SYSV	_M_SYSV
M_UNIX	_M_UNIX
M_WORDSWAP	_M_WORDSWAP
M_XENIX	_M_XENIX
M_XOUT	_M_XOUT
OS2	
_POSIX_SOURCE	
__STDC__	
_STRICT_NAMES	
_SVID	
unix	__unix
_XOPEN_SOURCE	

__STDC__ is defined to be 1 if **-ansi**, **-posix**, or **-xpg3** are specified. Otherwise, it is set to 0.

-unix

Generates SCO UNIX COFF files. This is the default behavior.

-V*string*

Allows you to place a text string in your object file. Typically, a copyright notice or program version number is specified here. ***string*** must be enclosed within double quotation marks if it contains blank spaces or quotation marks. A backslash character must also precede any quotation marks within the ***string***.

-w Prevents compiler warning messages from being issued. This is equivalent to "**-W0**".

-W*num*
Sets the output level for compiler warning messages.

- If *num* is "0", no warning messages are issued.
- If *num* is "1", most warning messages are issued (this is the default value).
- If *num* is "2", more warning messages are issued, including use of functions with no declared return type, failure to put **return** statements in functions with non-**void** return types, and data conversions that would cause loss of data or precision.
- If *num* is "3", all of the above warning messages are issued, along with warnings about any non-ANSI features and extended keywords and function calls before the appearance of function prototypes in the program.

This option does not affect compiler error message output.

-WX This option causes all warnings to be fatal. It can be used in combination with **-W0**, **-W1**, **-W2**, or **-W3**.

-xenix
For XENIX Cross Development. This option produces OMF programs using XENIX libraries and **#include** files. The resulting programs are compatible with XENIX 386 System V. If used in conjunction with **-M2**, they are compatible with both 286 and 386 System V.

User libraries for XENIX Cross Development must be in **ranlib**(XNX) format; that is, the first member must be named __.SYMDEF, which is a dictionary for the library.

-x2.3 This option is equivalent to the **-xenix** option, but it also includes the extended functions available with XENIX System V 386 release 2.3. When used with **-M2**, this option produces XENIX System V 286 2.3 compatible files

-xout
Like **-x2.3**, but allows access to system calls provided by SCO UNIX 3.2. Programs generated using this option may not run properly under XENIX.

-xpg3
Enforces strict XPG3 conformance. The language is restricted to ANSI specifications. The XPG3, POSIX and ANSI conformant C libraries are used in the link phase. When using this option, you don't need to link in the math library (with **-lm**), because the math routines are included in the conformant libraries and are automatically linked in. If you specify both **-xpg3** and **-lm**, you will be warned about undefined symbols. Do not use the **-ansi** or **-posix** flags with this flag.

-xpg3plus
closely represents a true X/Open conforming implementation with SCO-added value (implemented by way of the include files and the functions found in *libcE.a*). Much of the added value is nonconforming since it pollutes the user's name space.

-X Removes the standard directories from the list of directories to be searched for **#include** files.

-z Displays the various compiler passes and their arguments but does not execute them.

-Za Restricts the language to ANSI specifications. The standard C library is used in the link phase. This is equivalent to the **-strict** option. Use the **-ansi** flag instead if the ANSI-conformant library is required in the link phase. The **-Za** is obsolete, and may be removed in a future release.

-Zd Includes line number information in the object file.

-Ze Enables the **far**, **near**, **huge**, **pascal**, and **fortran** keywords. This is equivalent to the **-Me** option.

-Zg Generates function declarations from function definitions and writes declarations to standard output (omits p2, p3, and **ld**).

-Zi Includes information for the screen-oriented debuggers **dbxtra** and **Codeview** and the symbolic debugger (**sdb**) in the output file. This option is equivalent to the **-g** option.

-Zl Removes default library information from the object file.

-Zp*n* Packs structure members in memory. The argument ***n*** can be one of the following:

- **1** allocates alignment to 1
- **2** allocates alignment to 2 (default for 80286 programs).
- **4** allocates alignment to 4 (default for 80386 programs).

-Zs Performs syntax check only (omits p2, p3, and **ld**).

> **NOTE** Many options (or equivalent forms of these options) are passed to the linker as the last phase of compilation. The **-M** option with the **s**, **m**, and **l** configuration options are passed to specify memory requirements. The **-i**, **-F**, and **-p** options are passed to specify other characteristics of the final program.

Default options

cc reads */etc/default/cc* to obtain information about default options and libraries. The default file may contain lines beginning with the patterns **FLAGS=** and **LIBS=**. Any parameters following the **FLAGS=** pattern are treated by **cc** as if they had been specified at the start of the **cc** command line. Parameters following the **LIBS=** pattern are treated as if they had been specified at the end of the command line. This option is intended for, but not restricted to, the specification of additional libraries. **cc** always searches for a file in */etc/default* that matches the last component of the pathname by which **cc** was invoked. Thus by linking **cc** to several different names and invoking it by those names, different defaults can be selected.

An example */etc/default/cc* file follows:

```
FLAGS= -M2
LIBS= -lx
```

This invokes the large model versions of the compiler passes to generate 286 code with **far** and **near** keywords enabled, and includes *libx.a* on all links.

There is also an environment variable **CCFLAGS**, which can also be used to set additional **cc** options. These are processed by the compiler driver before any command-line arguments. The **CCFLAGS** variable allows you to override options which are set using the **FLAGS=** pattern, and may itself be overridden by options you specify on the command line.

Files

/bin/cc	Driver
/lib/p1, p2, p3	286 compiler passes
/lib/386/c_1p, p2_386, p3_386m	386 compiler passes
/lib/*.a, /usr/lib/*.a	Standard COFF and 286 OMF libraries
/lib/386/*.a	386 OMF libraries
/lib/cc.err	Error message file
/lib/cc.hlp	Help message file
/lib/*crt*.o	Startup code files
/etc/default/cc	Default options and libraries

Notes

Error messages are produced by the program that detects the error. These messages are usually produced by the C compiler, but may occasionally be produced by the assembler or the linker.

See also

ar(CP), **ld**(CP), **lint**(CP), **machine**(HW), **masm**(CP), **ranlib**(XNX),

Standards conformance

cc is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

cdc

change the delta commentary of an SCCS delta

Syntax

cdc -r*SID* [**-m**[*mrlist*]] [**-y**[*comment*]] *files*
cdc -r*SID* **-m**[*mrlist*] **-y**[*comment*] **-**

Description

The **cdc** command changes the delta commentary, for the *SID* (SCCS identification string) specified by the **-r** option, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the **delta**(CP) command's **-m** and **-y** options.

If a directory is named, **cdc** treats each file in the directory as if it had been specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of **-** is given, **cdc** reads standard input (see "Warnings" below) and each line of standard input is taken to be the name of an SCCS file to be processed.

All the described options apply independently to each named file:

-r*SID* Used to specify the SCCS identification (SID) string of a delta for which the delta commentary is to be changed.

-m*mrlist* If the SCCS file has the **v** flag set (see **admin**(CP)) then a list of MR numbers to be added and/or deleted in the delta commentary of the *SID* specified by the **-r** option may be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of **delta**(CP). In order to delete an MR, precede the MR number with the character "!" (see "Examples" below). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If **-m** is not used and the standard input is a terminal, the prompt "`MRs?`" is issued on standard output before standard input is read; if the standard input is not a terminal, no prompt is issued. The "`MRs?`" prompt always precedes the `comments?` prompt (see **-y** option).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the **v** flag has a value (see **admin**(CP)), it is taken to be the name of a program (or shell procedure) which validates MR numbers. If a non-zero exit status is returned from the MR number validation program, **cdc** terminates and the delta commentary remains unchanged.

-y[*comment*] Arbitrary text used to replace the comment(s) already existing for the delta specified by the **-r** option. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If **-y** is not specified and standard input is a terminal, the prompt `comments?` is issued on standard output before standard input is read; if standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

Simply stated, if you made the delta, or you own the file and directory, you can modify the delta commentary.

Examples

```
cdc -r1.6 -m"bl78-12 !bl77-54 bl79-01" -ytrouble s.file
```

adds bl78-12 and bl79-01 to the MR list, removes bl77-54 from the MR list, and adds the comment **trouble** to delta 1.6 of s.file.

The same thing can be done as follows:

```
cdc -r1.6 s.file
MRs? !bl77-54 bl78-12 bl79-01
comments? trouble
```

Files

x-file (see **delta**(CP))

z-file (see **delta**(CP))

Warnings

If SCCS file names are supplied to **cdc** via standard input (using **-** on the command line), then the **-m** and **-y** option must also be used.

See also

admin(CP), **delta**(CP), **get**(CP), **prs**(CP), **sccsfile**(FP)

cflow

generate C flowgraph

Syntax

cflow [**-d***num*] [**-ix**] [**-i_**] [**-n***function*] [**-r**] *files*

Description

The **cflow** command analyzes a collection of C, **yacc**, **lex**, assembler, and object files and attempts to build a graph charting the external references. Files suffixed with *.y*, *.l*, and *.c* are "yacc'ed," "lex'ed," and C-preprocessed as appropriate. Files suffixed with *.i* are processed by the first pass of **cc**. Files suffixed with *.s* are assembled. Assembled files, and files suffixed with *.o*, have information extracted from their symbol tables. The results are collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference number, followed by a number of tabs indicating the level, then the name of the global symbol followed by a colon and its definition. Normally only function names that do not begin with an underscore are listed (see the **-i** options below). For information extracted from C source files, the definition consists of an abstract type declaration (for example, ***char** **), and, delimited by angle brackets, the name of the source file and the line number where the definition was found.

Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (for example, **text**). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only <> is printed.

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}

f()
{
        i = h();
}
```

the command

cflow -ix file.c

produces the output

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the output of **cflow** can be piped to **pr**(C), using the **-e** option, to compress tab expansion to less than eight spaces.

In addition to the **-D**, **-I**, and **-U** options (which are interpreted just as they are by **cc**(CP) and **cpp**(CP)), the following options are interpreted by **cflow**:

-d*num* The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be ignored.

-ix Include external and static data symbols. The default is to include only functions in the flowgraph.

-i_ Include names that begin with an underscore. The default is to exclude these functions (and data if **-ix** is used).

-n*function*
Generate a flowgraph from *function* on to the end of the file.

-r Reverse the "caller:callee" relationship, producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

Warning

This utility does not support code containing function prototypes.

Diagnostics

cflow complains about bad options. It also complains about multiple definitions and only accepts the first one. Other messages may come from the various programs used (for example, the C-preprocessor).

Notes

Files produced by **lex**(CP) and **yacc**(CP) cause the reordering of line number declarations which can confuse **cflow**. To get proper results, feed **cflow** the **yacc** or **lex** input.

See also

as(CP), **cc**(CP), **cpp**(CP), **lex**(CP), **nm**(CP), **pr**(C), **yacc**(CP)

Standards conformance

cflow is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

chkshlib

compare shared libraries tool

Syntax

chkshlib [-b] [-i] [-n] [-v] *file1* [*file2 file3* ...]

Description

The **chkshlib** command checks for compatibility between files. Input files can be combinations of host shared libraries, non-stripped target shared libraries, and non-stripped executable files. A file is compatible with another file if every library symbol in it that should be matched is matched in the second (that is, the symbol exists and has the same address in both files). The path name for the target shared library in both files must be identical (unless the **-i** option is set).

It is possible for *file1* to be compatible with *file2* without the reverse also being true.

If one incompatibility is found it is reported to standard output and processing stops (unless the **-v** option is set).

The options to **chkshlib** are:

-b If there are symbols found in *file1* that are not in the bounds of *file2*, report warning messages to standard output.

-i Turn off the restriction that the path names for the target shared library need to be identical for two files to be compatible.

-n Indicate that there are exactly two input files, which are target shared libraries, where the first references symbols in the second ("includes" the second).

-v Cause verbose reporting of all incompatibilities to standard output.

The output of **chkshlib** depends upon the input. If the first input file is an executable file and the other input files, if any, are target shared libraries, the output states whether or not the executable file can execute using each target shared library. If there are no target shared libraries supplied, **chkshlib** performs the compatibility check against the target shared libraries specified in the *.lib* section of the executable file.

If the first input file is an executable file and the other input file(s) is a host shared library, the output states whether or not the executable file could have been produced using each host.

If one input file is a host shared library and the other input file, if any, is a target shared library, the output states whether or not the host shared library could produce executable files that will run with the target shared library. If no target shared library is supplied, then **chkshlib** performs the compatibility check against the target specified in the *.lib* section of the library definition file found in the host.

If both input files are target shared libraries or both input files are host shared libraries, the output states whether or not the first file could replace the second and vice versa.

If both input files are target libraries and the **-n** option is set, the output states if the first file references symbols in the second file ("includes" the second).

Compatibility of all other combinations of host shared libraries, target shared libraries, and executable files has no useful meaning, and these other combinations of files are not accepted as valid input to **chkshlib**.

The valid combinations of input files are:

Executable	[Target shared library(1) ... Target shared library(n)]
Executable	[Host shared library(1) ... Host shared library(n)]
Host shared library	Target shared library
Host shared library	Host shared library
Target shared library	Host shared library
Target shared library	Target shared library

All other forms of input will be rejected.

Diagnostics

Exit status is 0 if no incompatibilities are found, 1 if an incompatibility is found, and 2 if a processing error occurs.

Notes

chkshlib requires that you use the **-i** option whenever you use the **-n** option.

Standard binaries distributed with the system are stripped, and **chkshlib** cannot be used with them.

See also

mkshlib(CP)

codeview

visual debugger

codeview - visual debugger

cv - visual debugger

Syntax

cv [*options*] [*executable_file* [*arguments*]]

Description

Codeview is a screen-oriented debugger which operates in "window mode" on either serial lines or the system console. **Codeview**'s pull-down menus and simple keystrokes offer fast access to most common commands.

In addition to numerous standard debugging functions, **Codeview** permits you to browse source or assembly code, indicates which line will be executed next, highlights breakpoints, switches screens to display program output, watches the value of variables (local or global), offers fast breakpoints using hardware debug registers, and provides a variety of convenient ways for examining and modifying process text and data. **Codeview** can debug programs written in C, FORTRAN, BASIC, or Pascal as well as assembly language. **Codeview** can debug both COFF and *x.out* executable files. **Codeview** provides support for interpreting core files and extends full control of process signals to the user.

To full take advantage of **Codeview**'s debugging features, the executable file should be compiled and linked with the correct options for generating symbolic debug and line number records (use the **-g** or **-Zd** and **-Zi** options when compiling with **cc**). The executable should at least have a standard symbol table (the executable should not be stripped). Otherwise, **Codeview** can only provide assembly level debugging, without symbol names. It is possible for only a portion (a single module) of the executable to be compiled with symbolic debug support. The correct options for compiling and linking for use with **Codeview** are described in the volume titled *Debugging Tools Guide*.

Codeview looks for default files named *a.out* and *core* in the current directory, and uses them as the executable and core files.

Due to implementation changes, CodeView now starts up the child process under a vt100 emulator. As a consequence, Curses-based programs do not always behave in the same way when run under CodeView as when run independently of the debugger.

Any command line parameters following the executable file name will be passed as command line arguments to the process. Once **Codeview** is running, arguments can be reset using l (restart).

Codeview takes the following options:

-c ***command-list***
Execute ***command-list*** on startup.

-C ***corefile*** | **-**
Load ***corefile*** instead of the default *core* in the current directory. Use **-C -** to ignore *core*.

-I ***dir*** Include ***dir*** in the default list of directories to be searched while looking for a source file.

-r Use 80386 hardware debug registers for breakpoints set in data.

-tsf Ignore statefile.

Command syntax

Dialog commands use the following syntax:

***command*[*arguments*][;*command*]**

Dialog commands are not case sensitive. Symbol names may be case sensitive.

Many **Codeview** commands take addresses or address ranges as arguments. For most commands, the following address constructs are valid:

Address	Constructs
symbol	symbolname
register	[@]register
address	[segment:]offset
range	startaddress endaddress
	startaddress L count
	startaddress (uses a default count, usually 128)
line number	.[filename:]linenumber

Several **Codeview** commands take expressions as arguments. Expressions can be complex C expressions using constants, symbols, operators, casts and function calls. Expression evaluation corresponds to that of the C compiler including precedence and type conversions.

Constants use the following syntax:

Constant	Syntax
digits	default radix
0digits	octal
0xdigits	hexadecimal
0ndigits	decimal
"chars"	null terminated string

Registers may also be referred to directly for reference, addressing and assignment using standard names (such as *"ds," "ebp,"* and *"ah"*). If a symbol uses the same name as a register, the symbol name overrides the register name. In this case, the register can be specified using the "@" symbol (for example: *@ds*).

The following assembly style directives can be used with register names for addressing instead of brackets:

BY byte pointer (pointer to 8 bits)

WO word pointer (pointer to 16 bits)

DW double word pointer (pointer to 32 bits)

For example, **DW esi+ebp+6** is equivalent to **DWORD PTR [esi][ebp+6]**.

If CodeView is running on the system console, these keys, in conjunction with other keys, provide access to CodeView functions. To access the ⟨Alt⟩, ⟨Shift⟩, and ⟨Ctrl⟩ key functions while running CodeView on a serial line, pseudo tty, **Xterm**, or **scoterm**, use the following equivalent combinations:

For	Use
⟨Alt⟩*Key*	⟨Ctrl⟩A followed by *Key*
⟨Shift⟩*Key*	⟨Ctrl⟩Z followed by *Key*
⟨Ctrl⟩*Key*	⟨Ctrl⟩B followed by *Key*
⟨Ctrl⟩⟨Shift⟩*Key*	⟨Ctrl⟩B⟨Ctrl⟩Z followed by *Key*

Pull Down Menus	
File	
Open Source...	Opens a source file
Open Module...	Opens a module from a module group
Print...	Prints a file
Unix Shell	Forks a child shell
Exit	Returns to operating system
Edit	
Undo ⟨Alt⟩+⟨Bksp⟩	Undoes most recent edit
Copy ⟨Ctrl⟩+⟨Ins⟩	Copies selected text to Clipboard
Paste ⟨Shift⟩+⟨Ins⟩	Inserts text from Clipboard
View	
Source	Opens a new source window
Memory	Opens a new memory window
Register ⟨F2⟩	Toggles register window
8087	Toggles 8087 window
Local	Toggles local window
Watch	Toggles watch window
Command	Toggles command window

(Continued on next page)

(Continued)

Pull Down Menus	
Help	Toggles help window
Output ⟨F4⟩	Displays program output
Maximize ⟨Ctrl⟩+⟨F10⟩	Toggles current window to full size
Size ⟨Ctrl⟩+⟨F8⟩	Resizes current window
Close ⟨Ctrl⟩+⟨F4⟩	Closes current window
Search	
Find...	Searches for expression
Selected Text ⟨Ctrl⟩+	Searches for selected text
Repeat Last Find ⟨Alt⟩+/	Repeats last Find command
Label/Function...	Searches for label or function
Run	
Restart	Reloads program into memory
Animate	Traces code, highlighting lines
Set Runtime Arguments...	Sets command-line arguments
Watch	
Add Watch... ⟨Ctrl⟩+W	Sets watch expression
Delete Watch... ⟨Ctrl⟩+U	Removes watch expression
Set Breakpoint... ⟨F9⟩	Sets breakpoints
Edit Breakpoints...	Edits/removes breakpoints
Quick Watch... ⟨Shift⟩+⟨F9⟩	Evaluates expression
Options	
Source Window...	Source-window display
Memory Window...	Memory-window display
Trace Speed...	Program-animation speed
Language...	Expression evaluation language
Screen Swap	Screen flipping/swapping
Ignore Signals	Ignore signals from the child
Emulator Scroll	VT100 emulator for child scrolls
Case Sensitivity	Case sensitivity of symbols
Calls	
Stack Backtrace	
Misc	
Next Window ⟨F6⟩	Next window
Previous Window ⟨Shift⟩+⟨F6⟩	Previous window
Execute to Cursor ⟨F7⟩	Execute to cursor
Memory Mode ⟨Shift⟩+⟨F3⟩	Memory-window display mode
Help	
Index	Alphabetic index of topics
Contents ⟨Shift⟩+⟨F1⟩	Table of contents
Topic ⟨F1⟩	Help on specific topic
Help On Help	Help on using the help system

In full screen mode, function keys perform the following operations:

⟨F1⟩ Help
⟨F2⟩ Toggle register window
⟨F3⟩ Switch between source, assembly and mixed mode
⟨F4⟩ Toggle between **Codeview** screen and application output
⟨F5⟩ Go
⟨F6⟩ Switch to next window
⟨F7⟩ Go to cursored line
⟨F8⟩ Trace (for example, into functions)
⟨F9⟩ Toggle breakpoint at cursored line
⟨F10⟩ Step (for example, over functions)

Code display

The following table lists the commands that are used to view code:

Code	Display
set source mode	s [+|-|&]
	+ C source
	- assembly
	& mixed
	with no operand, displays current mode
view source	v [address]
search for regular expression	/regular expression
Move in the current window	⟨PgUp⟩, ⟨PgDn⟩
	arrow keys
	⟨Home⟩, ⟨End⟩
unassemble code	U [address | range]

Data display

The following commands display data:

?*expression*[,*format*]
This table lists the valid format specifiers:

Format	Specifiers
d,i	signed decimal integer (16 bit)
u	unsigned decimal integer (16 bit)
o	unsigned octal integer (16 bit)
x,X	unsigned hexadecimal integer (16 bit)
f	floating point
e,E	scientific notation floating point
g,G	scientific notation floating point
c	character
s	null terminated string
h (16 bit)	used with d,o,u,x and X. Example: *,hd* (short)
l (32 bit)	used with d,o,u,x and X. Example: *,ld* (long)

??*expression*
The following special keystrokes are recognized:

⟨Return⟩ de-reference current object
⟨Esc⟩ return to dialog line

r [*register*]
This causes a register dump; without a register name, it shows all registers and disassembles the current instruction at cs:ip.

7 Math co-processor register dump.

Variable and memory assignment

Use the following commands to assign values to variables or registers:

?*expression* = *expression*
Left ***expression*** must evaluate to an value.

r *register* [*value*]
Assign value to indicated register. If ***value*** is omitted, prompt for a new value.

Special commands

The **D** (Dump) command displays memory contents, and uses this syntax:

D[*type*] [*address* | *range*]

The following types are recognized:

A	ASCII characters
B	bytes
D	four-byte words, hexadecimal
I	two-byte integers
J	four-byte integers
L	eight-byte long reals
S	four-byte short reals
T	ten-byte reals
U	unsigned two-byte integers
V	unsigned four-byte integers
W	two-byte words, hexadecimal

The **C** (Compare) command compares two memory regions and lists the differences. Use this syntax:

C *range1 address2*

The **S** (Search) command searches a memory range for a pattern, and lists the occurrences of the pattern. Use this syntax:

S *range list*

The **E** (Enter) command modifies memory contents, and uses this syntax:

E [*type*] *address* [*list*]

The **F** (Fill) command sets memory contents to and iterated pattern, and uses this syntax:

F *range list*

The **M** (Move) command copies memory contents from one region to another, and uses this syntax:

M *range1 address2*

The **A** (Assemble) modifies a text range using a dynamic assembler. This is the syntax:

A [*address*]

Execution control

The following commands control program execution:

g [*address*]
: Resume execution (go). If provided, go to *address*.

t [*count*]
: Trace; single-step; step into function on function call.

p [*count*]
Program; single-step; step over function calls (for example, atomic).

e "Slow motion" execute.

l [*args*]
Restart (load). Uses command line arguments if specified. Obeys **sh**(C) quoting conventions.

sig [*signo*]
Set pending signal; signal will take effect when execution is resumed with **g**, **t** or **p**. Without *signo*, display currently pending signal.

k Stack trace.

bp [*address* [,*passcount*] [,*commands*]]
set a breakpoint

bc [*list* | *]
clear breakpoint(s)

bl list breakpoints

be [*list* | *]
enable breakpoint(s)

bd [*list* | *]
disable breakpoint(s)

Watch Expressions

w? *expression*[,*format*]	watch expression
w	watch list
y [*list* | *]	watch delete

Redirection commands

< *filename*
Re-direct **Codeview** input from *filename*.

[*T*] *filename*
Re-direct **Codeview** output to *filename*. Optional argument *T* echoes output to **Codeview** screen as well as *filename*.

= *filename*
Re-direct input and output.

***** *comment*
Echo *comment* text to **Codeview** screen.

: Delay execution of commands by approximately 1/2 second.

" Pause until keystroke.

Miscellaneous commands

q	exit **Codeview**
![*shell command(s)*]	shell escape
@	redraw **Codeview** screen
h	help
⟨**Esc**⟩	clear command line; cancel menu command
n[*radix*]	set default input/output radix (8, 10 or 16) with no argument, shows current default radix
o[b | c | f][+ | -]	turn on or off or display current status of the following: **b** - bytes coded **c** - case sensitivity **f** - flip/swap (for example, save output)
****	screen exchange; switch to output screen
#number	tab set; specify width for tab to space expansion

Key bindings by key

Keys	Function
!	Shell escape
"	Pause
#	Tab set
*	Comment
.	Current location
/	Search
7	Display 8087 registers
:	Delay
<	Command window input redirection
=	Command window redirection reset
>	Command window output redirection
?	Display expression
??	Quick watch
@	Redraw
A	Assemble
⟨Alt⟩-*	Show menu corresponding to first letter
⟨Alt⟩-/	Repeat last find
⟨Alt⟩-⟨Bksp⟩	Undo last edit operation
⟨Alt⟩-⟨Ctrl⟩-F*	Virtual screen switch
⟨Alt⟩-⟨F1⟩	Previous help screen
BC	Breakpoint clear
BD	Breakpoint disable
BE	Breakpoint enable
⟨Bksp⟩	Delete character to left of cursor
BL	Breakpoint list
BP	Breakpoint set
C	Compare memory
⟨Ctrl⟩-A	Cursor word left
⟨Ctrl⟩-D	Cursor character right
⟨Ctrl⟩-⟨DOWN⟩	Scroll line down
⟨Ctrl⟩-⟨End⟩	Cursor EOF
⟨Ctrl⟩-F	Cursor word right
⟨Ctrl⟩-⟨F1⟩	Next help topic
⟨Ctrl⟩-⟨F10⟩	Maximize window
⟨Ctrl⟩-⟨F4⟩	Close window
⟨Ctrl⟩-⟨F8⟩	Resize window
⟨Ctrl⟩-G	Delete character at cursor
⟨Ctrl⟩-H	Delete character to left of cursor
⟨Ctrl⟩-⟨Home⟩	Cursor BOF
⟨Ctrl⟩-⟨Ins⟩	Copy to clipboard
⟨Ctrl⟩-⟨LEFT⟩	Cursor word left

(Continued on next page)

(Continued)

Keys	Function
⟨Ctrl⟩-⟨PgDn⟩	Scroll page right
⟨Ctrl⟩-⟨PgUp⟩	Scroll page left
⟨Ctrl⟩-⟨RIGHT⟩	Cursor word right
⟨Ctrl⟩-⟨Shift⟩-⟨End⟩	Select to EOF
⟨Ctrl⟩-⟨Shift⟩-⟨F1⟩	Previous help topic
⟨Ctrl⟩-⟨Shift⟩-⟨Home⟩	Select to BOF
⟨Ctrl⟩-⟨Shift⟩-⟨LEFT⟩	Select word to left
⟨Ctrl⟩-⟨Shift⟩-⟨RIGHT⟩	Select word to right
⟨Ctrl⟩-T	Delete rest of word on cursor
⟨Ctrl⟩-U	Delete watch expression
⟨Ctrl⟩-⟨UP⟩	Scroll line up
⟨Ctrl⟩-V	Toggle keyboard insertion mode
⟨Ctrl⟩-W	Add watch expression
⟨Ctrl⟩-X	Cursor line down
⟨Ctrl⟩-\	Find selected text
D	Dump memory
⟨Del⟩	Delete character at cursor
⟨DOWN⟩	Cursor line down
E	Enter memory
E	Execute
⟨End⟩	Cursor EOL
F	Fill memory
⟨F1⟩	Help
⟨F2⟩	Toggle register window
⟨F3⟩	Toggle source window display modes
⟨F4⟩	Toggle output and CV screen
⟨F5⟩	Run
⟨F6⟩	Next window
⟨F7⟩	Execute to cursor
⟨F8⟩	Trace into
⟨F9⟩	Toggle breakpoint
⟨F10⟩	Step over
G	Go
H	Help
⟨Home⟩	Cursor BOL
I	Port input
⟨Ins⟩	Toggle keyboard insertion mode
K	Stack trace
L	Load
⟨LEFT⟩ Cursor	character left
M	Move memory
N	Radix
O	Option set/display
O	Port output

(Continued on next page)

(Continued)

Keys	Function
P	Program step
⟨PgDn⟩	Scroll page down
⟨PgUp⟩	Scroll page up
Q	Quit
R	Register
⟨RIGHT⟩	Cursor character right
S	Search memory
S	Source display mode
⟨Shift⟩-⟨DOWN⟩	Select current line
⟨Shift⟩-⟨F1⟩	Contents
⟨Shift⟩-⟨F10⟩	Next history
⟨Shift⟩-⟨F3⟩	Toggle memory window display modes
⟨Shift⟩-⟨F6⟩	Previous window
⟨Shift⟩-⟨F8⟩	Previous history
⟨Shift⟩-⟨F9⟩	Open QuickWatch window for a variable
⟨Shift⟩-⟨Ins⟩	Insert from clipboard
⟨Shift⟩-⟨LEFT⟩	Select character to left
⟨Shift⟩-⟨PgDn⟩	Select screen following
⟨Shift⟩-⟨PgUp⟩	Select screen previous
⟨Shift⟩-⟨RIGHT⟩	Select character to right
⟨Shift⟩-⟨Tab⟩	Move to previous command
⟨Shift⟩-⟨UP⟩	Select previous line
T	Command window tee redirection
T	Trace
⟨Tab⟩	Move to next command
U	Unassemble
⟨UP⟩	Cursor line up
V	View source lines
W	List watch
W?	Add watch

Key bindings by function

Function	Keys
Add watch expression	⟨Ctrl⟩-W
Add watch	W?
Assemble	A
Breakpoint clear	BC
Breakpoint disable	BD
Breakpoint enable	BE
Breakpoint list	BL
Breakpoint set	BP
Close window	⟨Ctrl⟩-⟨F4⟩
Command window input redirection	<
Command window redirection reset	=
Command window output redirection	>
Command window tee redirection	T
Comment	*
Compare memory	C
Contents	⟨Shift⟩-⟨F1⟩
Copy to clipboard	⟨Ctrl⟩-⟨Ins⟩
Current location	.
Cursor word left	⟨Ctrl⟩-A
Cursor character right	⟨Ctrl⟩-D
Cursor EOF	⟨Ctrl⟩-⟨End⟩
Cursor word right	⟨Ctrl⟩-F
Cursor BOF	⟨Ctrl⟩-⟨Home⟩
Cursor word left	⟨Ctrl⟩-⟨LEFT⟩
Cursor word right	⟨Ctrl⟩-⟨RIGHT⟩
Cursor line down	⟨Ctrl⟩-X
Cursor line down	⟨DOWN⟩
Cursor EOL	⟨End⟩
Cursor BOL	⟨Home⟩
Cursor character left	⟨LEFT⟩
Cursor character right	⟨RIGHT⟩
Cursor line up	⟨UP⟩
Delay	:
Delete character to left of cursor	⟨Bksp⟩
Delete character at cursor	⟨Ctrl⟩-G
Delete character to left of cursor	⟨Ctrl⟩-H
Delete rest of word on cursor	⟨Ctrl⟩-T
Delete watch expression	⟨Ctrl⟩-U
Delete character at cursor	⟨Del⟩
Display 8087 registers	7
Display expression	?

(Continued on next page)

(Continued)

Function	Keys
Dump memory	D
Enter memory	E
Examine symbols	X
Execute	E
Execute to cursor	⟨F7⟩
Fill memory	F
Find selected text	⟨Ctrl⟩-\
Go	G
Help	⟨F1⟩
Help	H
Insert from clipboard	⟨Shift⟩-⟨Ins⟩
List watch	W
Load	L
Maximize window	⟨Ctrl⟩-⟨F10⟩
Move memory	M
Move to previous command	⟨Shift⟩-⟨Tab⟩
Move to next command	⟨Tab⟩
Next help topic	⟨Ctrl⟩-⟨F1⟩
Next window	⟨F6⟩
Next history	⟨Shift⟩-⟨F10⟩
Open QuickWatch window for a variable	⟨Shift⟩-⟨F9⟩
Option set/display	O
Pause	"
Port input	I
Port output	O
Previous help screen	⟨Alt⟩-⟨F1⟩
Previous help topic	⟨Ctrl⟩-⟨Shift⟩-⟨F1⟩
Previous window	⟨Shift⟩-⟨F6⟩
Previous history	⟨Shift⟩-⟨F8⟩
Program step	P
Quick watch	??
Quit	Q
Radix	N
Redraw	@
Register	R
Repeat last find	⟨Alt⟩-/
Resize window	⟨Ctrl⟩-⟨F8⟩
Run	⟨F5⟩
Screen exchange	\
Scroll line down	⟨Ctrl⟩-⟨DOWN⟩
Scroll page right	⟨Ctrl⟩-⟨PgDn⟩
Scroll page left	⟨Ctrl⟩-⟨PgUp⟩
Scroll line up	⟨Ctrl⟩-⟨UP⟩
Scroll page down	⟨PgDn⟩

(Continued on next page)

(Continued)

Function	Keys
Scroll page up	⟨PgUp⟩
Search	/
Search memory	S
Select to EOF	⟨Ctrl⟩-⟨Shift⟩-⟨End⟩
Select to BOF	⟨Ctrl⟩-⟨Shift⟩-⟨Home⟩
Select word to left	⟨Ctrl⟩-⟨Shift⟩-⟨LEFT⟩
Select word to right	⟨Ctrl⟩-⟨Shift⟩-⟨RIGHT⟩
Select current line	⟨Shift⟩-⟨DOWN⟩
Select character to left	⟨Shift⟩-⟨LEFT⟩
Select screen following	⟨Shift⟩-⟨PgDn⟩
Select screen previous	⟨Shift⟩-⟨PgUp⟩
Select character to right	⟨Shift⟩-⟨RIGHT⟩
Select previous line	⟨Shift⟩-⟨UP⟩
Shell escape	!
Show menu corresponding to first letter	⟨Alt⟩-*
Source display mode	S
Stack trace	K
Step over	⟨F10⟩
Tab set	#
Toggle keyboard insertion mode	⟨Ctrl⟩-V
Toggle register window	⟨F2⟩
Toggle source window display modes	⟨F3⟩
Toggle output and CV screen	⟨F4⟩
Toggle breakpoint	⟨F9⟩
Toggle keyboard insertion mode	⟨Ins⟩
Toggle memory window display modes	⟨Shift⟩-⟨F3⟩
Trace into	⟨F8⟩
Trace	T
Unassemble	U
Undo last edit operation	⟨Alt⟩-⟨Bksp⟩
View source lines	V
Virtual screen switch	⟨Alt⟩-⟨Ctrl⟩-F*
Yank watch	Y

Files

a.out
core
/usr/bin/cvtcoff
/usr/lib/codeview/cv.hlp
/usr/lib/codeview/pwb.hlp

Notes

Codeview executes transparently on either COFF or *x.out* binaries. The symbolic debug records from COFF binaries are translated to corresponding *x.out* format at startup using **cvtcoff**. This converter must be installed for debugging COFF binaries.

Standards conformance

Codeview is an extension of AT&T System V provided by the Santa Cruz Operation.

See also

a.out(FP), **adb**(CP), **cc**(CP), **core**(F), **ld**(CP), **sdb**(CP), **sh**(C), **syms**(FP)

comb

combine SCCS deltas

Syntax

comb [**-o**] [**-s**] [**-p***SID*] [**-***clist*] *files*
comb -

Description

The **comb** command generates a shell procedure (see **sh**(CP)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all arguments apply to all named SCCS files. If a directory is named, **comb** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of **-** is given, the standard input is read; each line of the input is taken to be the name of an SCCS file to be processed. The generated shell procedure is written to standard output.

The arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any argument apply independently to each named file.

-o For each **get -e** generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the **-o** may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

-s This argument causes **comb** to generate a shell procedure that, when run, produces a report giving the following for each file: the file name, the percentage change in file size after combining, the total file size (in blocks) after combining, and the original file size (also in blocks). The format of the report is as follows:

```
filename [±]percentage_change newsize/oldsize
```

A sample command line to use this option would look like this:

```
comb -s s.filename | sh > report
```

where *report* is the name of the file that receives the report information.

It is recommended that before any SCCS files are actually combined, you should use this option to determine exactly how much space can be saved through the combining process.

-p*SID* The *SID* (SCCS identification string) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file. For example, if you give the command:

```
comb -p 1.5 s.filename
```

All deltas from 1.5 to the most recent delta will be preserved. All deltas prior to 1.5 will be removed.

-c*list* This command causes the existing deltas in the subject file to be combined. It then overwrites the subject file with the new combined delta. For example, if you give the following command:

```
comb -c1.4 s.workfile | sh
```

all previous deltas to *s.workfile* will be combined, and the new combined delta will be number 1.4.

If no arguments are specified, **comb** will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

Files

s.COMB	The name of the reconstructed SCCS file.
comb?????	Temporary file.

Notes

The **comb** command may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

See also

admin(CP), **delta**(CP), **get**(CP), **prs**(CP), **sccsfile**(FP), **sh**(C)

conv

common object file converter

Syntax

conv [**-a**] [**-o**] [**-p**] [**-s**] **-t** *target* [**-** | *files*]

Description

The **conv** command converts object files in the Common Object File Format (COFF) from their current byte ordering to the byte ordering of the *target* machine. The converted file is written to *file.v*. **conv** can be used on either the source (sending) or target (receiving) machine.

Command line options are:

- Indicates that the names of *files* should be read from standard input.

-a If the input file is an archive, produce the output file in the System V Release 2.0 portable archive format.

-o If the input file is an archive, produce the output file in the old (pre-System V) archive format.

-p If the input file is an archive, produce the output file in the System V Release 1.0 random access archive format.

-s "Preswab" all characters in the object file. This is useful only for 3B20 computer object files which are to be "swab-dumped" from a DEC machine to a 3B20 computer.

-t *target* Convert the object file to the byte ordering of the machine (*target*) to which the object file is being shipped. This may be another host or a target machine. Legal values for *target* are: pdp, vax, ibm, x86, b16, n3b, mc68, and m32.

The **conv** command is meant to ease the problems created by a multi-host, cross-compilation development environment. The **conv** command is best used within a procedure for shipping object files from one machine to another.

The **conv** command will recognize and produce archive files in three formats: the pre-System V format, the System V Release 1.0 random access format, and the System V Release 2.0 portable ASCII format. By default, **conv** will create the output archive file in the same format as the input file. To produce an output file in a different format than the input file, use the **-a**, **-o**, or **-p** options. If the output archive format is the same as the input format, the archive symbol table will be converted, otherwise the symbol table will be stripped from the

archive. The **ar**(CP) command with its **-t** and **-s** options must be used on the target machine to recreate the archive symbol table.

Example

To ship object files from a VAX computer system to a 3B2 computer, execute the following commands:

```
conv -t m32 *.out
uucp *.out.v my3b2!~/rje/
```

Diagnostics

The diagnostics are self-explanatory. Fatal diagnostics on the command lines cause termination. Fatal diagnostics on an input file cause the program to continue to the next input file.

Notes

The **conv** command will not convert archives from one format to another if both the source and target machines have the same byte ordering. The system tool **convert**(CP) should be used for this purpose.

See also

a.out(FP), **ar**(CP), **ar**(FP), **convert**(CP)

Standards conformance

conv is conformant with:

AT&T SVID Issue 2.

convert

convert archive files to common formats

Syntax

convert [**-x**] [**-5**] *infile outfile*

Description

The **convert** command transforms input *infile* to output *outfile*. *infile* must be an archive file and *outfile* will be the equivalent System V archive file. All other types of input to the **convert** command will be passed unmodified from the input file to the output file (along with appropriate warning messages).

The **-x** option is required to convert a XENIX archive. Using this option will convert the general archive structure but leave archive members unmodified.

The **-5** option is used to convert a non-System V archive file.

infile must be different from *outfile*.

Files

TMPDIR/conv*	Temporary files.
	TMPDIR is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**( ) in **tmpnam**(S)).

See also

a.out(FP), **ar**(CP), **ar**(FP), **tmpnam**(S)

cpp

the AT&T C language preprocessor

Syntax

cpp [*options*] [*ifile* [*ofile*]]

Description

The C language preprocessor, **cpp**, is invoked as the first pass of any C compilation by the **rcc** command. **cpp**'s output is a form acceptable as input to the next pass of the C compiler. The use of **cpp** other than through the **rcc** command is not suggested. See **m4**(CP) for a description of a general macro processor.

The **cpp** command optionally accepts two file names as arguments. *ifile* and *ofile* are, respectively, the input and output for the preprocessor. They default to standard input and standard output.

Options

-C By default, **cpp** strips C-style comments. If the **-C** option is specified, all comments (except those found on **cpp** directive lines) are passed along.

-D*name*[=*def*]
Define *name* with value *def* as if by a **#define**. If no =*def* is given, *name* is defined with value 1. The **-D** option has lower precedence than the **-U** option. That is, if the same name is used in both a **-U** option and a **-D** option, the name will be undefined regardless of the order of the options.

-H Print the path names of included files, one per line on standard error.

-I*dir* Search in *dir* for **#include** files whose names do not begin with " / " before looking in the directories on the standard list. **#include** files whose names are enclosed in double quotes and do not begin with a " / " will be searched for first in the current directory, then in directories named on **-I** options, and last in directories on the standard list.

-P Preprocess the input without producing the line control information used by the next pass of the C compiler.

-S*n* This option allows you to set the size of the "sbf" buffer. *n* is the number of bytes that will be dynamically allocated to the "sbf" buffer. The "sbf" buffer is used, among other things, to store the values of **#defines**. If you run out of space because of a large number of

#defines, you can use the **-S** option to increase the buffer size. If the **-S** option is not used, a default statically allocated buffer is set up. If you specify some illegal value for **n** then the minimum allowable value will be printed in a warning message.

-T The **-T** option forces **cpp** to use only the first eight characters to distinguish preprocessor symbols and is included for backward compatibility.

-U*name*
Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the preprocessor, or a name defined on a **-D** option. The names predefined by **cpp** are **unix** and **i386**.

-Y*dir*
Use directory *dir* in place of the standard list of directories when searching for **#include** files.

Special names

Two special names are understood by **cpp**. The name **__LINE__** is defined as the current source line number (as a decimal integer), and **__FILE__** is defined as the current file name (as a C string). They can be used anywhere (including in macros) just like any other defined names. **cpp**'s notion of the line number and filename may be changed using a **#line** directive (see below).

Directives

All **cpp** directive lines start with # in column 1. Any number of blanks and tabs is allowed between the # and the directive. The directives are:

#define *name token-string*
Defines a macro called *name*. The value of name is *token-string*. Subsequent instances of *name* are replaced with *token-string*.

#define *name*(*arg*, . . . , *arg*) *token-string*
This allows substitution of a macro with arguments. The token-string will be substituted for occurrences of *name* in the input file. If the occurrence of *name* is followed by an argument list (a comma-separated list of tokens, enclosed in parentheses), each token in the list replaces the corresponding *arg* in the macro definition. After the entire *token-string* has been expanded, **cpp** processes the text that was just substituted. Notice that there can be no space between *name* and the " (".

#undef *name*
Removes the definition of the macro *name*. No additional tokens are permitted on the directive line after *name*.

#ident "*string*"
Put *string* into the .comment section of an object file.

#include "*filename*", **#include** <*filename*>
Include the contents of *filename* at this point in the program. The included text will then itself be processed by **cpp**. **#include** files whose names are enclosed in <> are searched for only in the standard list of directories. See the **-I** and **-Y** options above for more detail. No additional tokens are permitted on the directive line after the final " or >.

#line *integer-constant* "*filename*"
Causes **cpp** to generate line control information for the next pass of the C compiler. The compiler will behave as if *integer-constant* is the line number of the next line of source code and *filename* (if present) is the name of the input file. No additional tokens are permitted on the directive line after the optional *filename*.

#endif
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. No additional tokens are permitted on the directive line.

#ifdef *name*
The lines following this directive and up to the matching **#endif** or next **#else** or **#elif** will appear in the output if *name* is currently defined. No additional tokens are permitted on the directive line after *name*.

#ifndef *name*
The lines following this directive and up to the matching **#endif** or next **#else** or **#elif** will appear in the output if *name* is not currently defined. No additional tokens are permitted on the directive line after *name*.

#if *constant-expression*
The lines following this directive and up to the matching **#endif** or next **#else** or **#elif** will appear in the output if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary –, !, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the use of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by **cpp** should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

To test whether either of two symbols, **foo** and **fum**, are defined, use

```
#if defined(foo) || defined(fum)
```

#elif ***constant-expression***
An arbitrary number of **#elif** directives is allowed between a **#if**, **#ifdef**, or **#ifndef** directive and a **#else** or **#endif** directive. The lines following the **#elif** and up to the next **#else**, **#elif**, or **#endif** directive will appear in the output if the preceding test directive evaluates to zero, all intervening **#elif** directives evaluate to zero, and the ***constant-expression*** evaluates to non-zero. If ***constant-expression*** evaluates to non-zero, all succeeding **#elif** and **#else** directives will be ignored. Any ***constant-expression*** allowed in a **#if** directive is allowed in a **#elif** directive.

#else
The lines following this directive and up to the matching **#endif** will appear in the output if the preceding test directive evaluates to zero, and all intervening **#elif** directives evaluate to zero. No additional tokens are permitted on the directive line. The test directives and the possible **#else** directives can be nested.

Files

INCDIR Standard directory list for **#include** files, usually */usr/include*.

LIBDIR Usually */lib*.

Diagnostics

The error messages produced by **cpp** are intended to be self-explanatory. The line number and file name where the error occurred are printed along with the diagnostic.

Notes

Because the standard directory for included files may be different in different environments, the following form of **#include** directive should be used:

```
#include <file.h>
```

This is preferred over one with an absolute path, such as the following:

```
#include "/usr/include/file.h"
```

The **cpp** command warns about the use of the absolute path name.

See also

cc(CP), **lint**(CP), **m4**(CP), **rcc**(CP), **rlint**(CP)

Standards conformance

cpp is conformant with:
AT&T SVID Issue 2.

cprs

compress a common object file

Syntax

cprs [-p] *infile outfile*

Description

The **cprs** command reduces the size of a common object file, *infile*, by removing duplicate structure and union descriptors. The reduced file, *outfile*, is produced as output.

The sole option to **cprs** is:

-p Print statistical messages including: total number of tags, total duplicate tags, and total reduction of *infile*.

Notes

Note that only COFF files are supported by this utility.

See also

a.out(FP), **strip**(CP), **syms**(FP)

cscope

interactively examine a C program

Syntax

cscope [**-f** *reffile*] [**-i** *namefile*] [**-I** *incdir*] [**-d**] [*files*]

Description

cscope is an interactive screen-oriented tool that helps programmers browse through C source code.

By default, **cscope** examines the C, **yacc**, and **lex** source files in the current directory and builds a symbol cross-reference. It then uses this table to find references to symbols (including C preprocessor symbols), function declarations, and function calls.

cscope builds the symbol cross-reference the first time it is used on the source files for the program being browsed. On a subsequent invocation, **cscope** rebuilds the cross-reference only if a source file has changed or the list of source files is different. When the cross-reference is rebuilt, the data for the unchanged files are copied from the old cross-reference, which makes rebuilding faster than the initial build.

The following options can appear in any combination:

-f *reffile*
: Use *reffile* as the cross-reference file name instead of the default *cscope.out*.

-i *namefile*
: Get the list of files (file names separated by spaces, tabs, or newlines) to browse from *namefile*. If this option is specified, **cscope** ignores any files appearing on the command line.

-I *incdir*
: Look in *incdir* (before looking in *INCDIR*, the standard place for header files that is normally */usr/include*) for any **#include** files whose names do not begin with / and that are not specified on the command line or in *namefile* above. (The **#include** files may be specified with either double quotes or angle brackets.) The *incdir* directory is searched in addition to the current directory (which is searched first) and the standard list (which is searched last). If more than one occurrence of **-I** appears, the directories are searched in the order they appear on the command line.

-d Do not update the cross-reference.

Requesting the initial search

After the cross-reference is ready, **cscope** will display this menu:

```
List references to this C symbol:
Edit this function or #define:
List functions called by this function:
List functions calling this function:
List lines containing this text string:
Change this text string:
```

Press the ⟨Tab⟩ or ⟨Return⟩ key repeatedly to move to the desired input field, type the text to search for, and then press the ⟨Return⟩ key.

Issuing subsequent requests

If the search is successful, any of these single-character commands can be used:

1-9	Edit the file referenced by the given line number.
⟨Space⟩	Display next lines.
+	Display next lines.
–	Display previous lines.
^e	Edit all lines.
>	Append the displayed list of lines to a file.

At any time these single-character commands can also be used:

⟨Tab⟩	Move to next input field.
⟨Return⟩	Move to next input field.
^m	Move to next input field.
^p	Move to previous input field.
.	Search with the last text typed.
^r	Rebuild the cross-reference.
!	Start an interactive shell (type **^d** to return to **cscope**).
^l	Redraw the screen.
?	Display this list of commands.
^d	Exit **cscope**.

NOTE If the first character of the text to be searched for matches one of the above commands, escape it by typing a " " (backslash) first.

Substituting new text for old text

After the text to be changed has been typed, **cscope** will prompt for the new text, and then it will display the lines containing the old text. Select the lines to be changed with these single-character commands:

1-9	Mark or unmark the line to be changed.
*****	Mark or unmark all displayed lines to be changed.
⟨Space⟩	Display next lines.
+	Display next lines.
–	Display previous lines.
a	Mark all lines to be changed.
^d	Change the marked lines and exit.
⟨Esc⟩	Exit without changing the marked lines.
!	Start an interactive shell (type **^d** to return to **cscope**).
^L	Redraw the screen.
?	Display this list of commands.

Environment variables

EDITOR	Preferred editor, which defaults to **vi**(C).
HOME	Home directory, which is automatically set at login.
SHELL	Preferred shell, which defaults to **sh**(C).
TERM	Terminal type, which must be a screen terminal.
VIEWER	Preferred file display program (such as **pg**(C)), which overrides **EDITOR** (see above).
VPATH	An ordered list of directory names, separated by colons. It is used by **cscope** to search for both source and header files. If **VPATH** is set, **cscope** searches for source files in the directories specified; if it is not set, **cscope** searches only in the current directory. **cscope** searches for header files in the following order: if **VPATH** is set, in directories specified in **VPATH** and if **VPATH** is not set, in the current directory, in directories specified by the **-I** option (if they exist), and in the standard location for header files (normally */usr/include*).

Files

cscope.out	Symbol cross-reference file, which is put in the home directory if it cannot be created in the current directory.
ncscope.out	Temporary file containing new cross-reference before it replaces the old cross-reference.
INCDIR	Standard directory for **#include** files (usually is */usr/include*).

Warnings

cscope recognizes function definitions of the form:

fname (args) arg_decs { . . . }

where:

fname	is the function name, followed by zero or more spaces or tabs (except a newline character).
args	is any string that does not contain a double-quotation (") or a newline, followed by zero or more spaces or tabs.
arg_decs	are zero or more argument declarations. *arg_decs* may include comments and white space.

It is not necessary for a function declaration to start at the beginning of a line. The return type may precede the function name; **cscope** will still recognize the declaration. Function definitions that deviate from this form will not be recognized by **cscope**.

ctrace

C program debugger

Syntax

ctrace [*options*] [*file*]

Description

The **ctrace** command allows you to follow the execution of a C program, statement-by-statement. The effect is similar to executing a shell procedure with the **-x** option. **ctrace** reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to standard output. You must put the output of **ctrace** into a temporary file because **cc**(CP) does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the **bfs**(C) or **tail**(C) commands.

The options commonly used are:

-f *functions* Trace only these *functions*.

-v *functions* Trace all but these *functions*.

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation.

String arguments to the **string**(S) functions and return values from **fgets**(S), **gets**(S), and **sprintf**(S) are printed as strings. You can request that variables be printed in additional formats, if appropriate, with these options:

-o Octal
-x Hexadecimal
-u Unsigned
-e Floating point

These options are used only in special circumstances:

-l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

-s Suppress redundant trace output from simple assignment statements and string copy function calls. T s option can hide a bug caused by use of the " = " operator in place of the " == " operator.

-t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The "Diagnostics" section explains when to use this option.

-P Run the C preprocessor on the input before tracing it. You can also use the **-D**, **-I**, and **-U** **cpp**(CP) options.

These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX type environment:

-b Use only basic functions in the trace code, that is, those in **ctype**(S), **printf**(S), and **string**(S). These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have **signal**(S), **fflush**(S), **longjmp**(S), or **setjmp**(S).

-p *string*
Change the trace print function from the default of **printf(**. For example, **fprintf(stderr,** would send the trace to standard error output.

-r *f* Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **-p** option).

Example

If the file *lc.c* contains this C program:

```
 1 #include <stdio.h>
 2 main()        /* count lines in input */
 3 {
 4         int c, nl;
 5
 6         nl = 0;
 7         while ((c = getchar()) != EOF)
 8                 if (c = '\n')
 9                         ++nl;
10         printf("%d\n", nl);
11 }
```

And you enter these commands and test data:

cc lc.c
a.out
1
⟨Ctrl⟩-d

the program will be compiled and executed. The output of the program will be the number **2**, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke **ctrace** with these commands:

ctrace lc.c >temp.c
cc temp.c
a.out

the output will be:

```
2 main()
6         nl = 0;
           /* nl == 0 */
7         while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
          /* c == 49 or '1' or "text" */
8                 if (c = '\n')
                   /* c == 10 or '\n' */
9                         ++nl;
                          /* nl == 1 */
7         while ((c = getchar()) != EOF)
          /* c == 10 or '\n' */
8                 if (c = '\n')
                   /* c == 10 or '\n' */
9                         ++nl;
                          /* nl == 2 */
7         /* repeating */
```

If you now enter an end-of-file character (⟨Ctrl⟩-D) the final output will be:

```
     /* repeated < 1 time */
 7     while ((c = getchar()) != EOF)
       /* c == -1 */
10     printf ("%d0, nl);
       /* nl == 2 */2
     /* return */
```

Note that the program output printed at the end of the trace line for the ***nl*** variable. Also note the **return** comment added by **ctrace** at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator ("=") in place of the equality operator ("=="). You can easily miss this error during code reading.

Execution-Time trace control

The default operation for **ctrace** is to trace the entire program file, unless you use the **-f** or **-v** options to trace specific functions. This does not give you statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding **ctroff()** and **ctron()** function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with **if** statements, and you can even conditionally include this code because **ctrace** defines the **CTRACE** preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from **sdb**(CP) if you compile with the **-g** option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctron()
r
```

You can also turn the trace off and on by setting static variable tr_ct_ to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly.

Files

/usr/lib/ctrace/runtime.c Run-time trace package.

Diagnostics

This section contains diagnostic messages from both **ctrace** and cc(CP), since the traced code often gets some **cc** warning messages. You can get **cc** error messages in some rare cases, all of which can be avoided.

ctrace diagnostics

Warning: some variables are not traced in this statement
Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the **-t** option to increase this number.

Warning: statement too long to trace
This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

Cannot handle preprocessor code, use -P option
This is usually caused by **#ifdef/#endif** preprocessor statements in the middle of a C statement, or by a semicolon at the end of a **#define** preprocessor statement.

'if ... else if' sequence too long
Split the sequence by removing an **else** from the middle.

Possible syntax error, try -P option
Use the **-P** option to preprocess the **ctrace** input, along with any appropriate **-D**, **-I**, and **-U** preprocessor options. If you still get the error message, check the "Warnings" section below.

cc diagnostics

Warning: illegal combination of pointer and integer
Warning: statement not reached
Warning: sizeof returns 0
Ignore these messages.

Compiler takes size of function
See the **ctrace** "possible syntax error" message above.

yacc stack overflow
See the **ctrace** "'if ... else if' sequence too long" message above.

Out of tree space; simplify expression
Use the **-t** option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

Redeclaration of signal
Either correct this declaration of **signal**(S), or remove it and **#include** *<signal.h>*.

Warnings

You will get a **ctrace** syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

The **ctrace** command assumes that BADMAG is a preprocessor macro, and that EOF and NULL are **#defined** constants. Declaring any of these to be variables, for example, "int EOF;", will cause a syntax error.

This utility does not support code containing function prototypes.

Notes

The **ctrace** command does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. **ctrace** may choose to print the address of an aggregate or use the wrong format (for example, 3.149050e-311 for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multifile program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

See also

bfs(C), **ctype**(S), **fclose**(S), **printf**(S), **setjmp**(S), **signal**(S), **string**(S), **tail**(C)

cxref

generate C program cross-reference

Syntax

cxref [*options*] *files*

Description

The **cxref** command analyzes a collection of C files and attempts to build a cross-reference table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or, with the **-c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

The default behavior of **cxref** is to cross-reference everything. You can also specify which items you want cross-referenced by using the **-i** option to turn off the default behavior, and one or more of **-a**, **-e**, **-f**, **-l**, **-m**, **-v**, and **-y**.

The following *options* are interpreted by **cxref**:

-A Turn off macro cross-referencing.

-c Print a combined cross-reference of all input files.

-D*name=def*
Define *name* with value *def* as if by a **#define**. If no *=def* is given, *name* is defined with value 1.

-E Turn off constant cross-references.

-F Turn off function cross-references.

-i Turn off all cross-referencing, except that which is specified using the following flags (**-i** *must* be specified first on the command line):

- **-a** Turn on macro cross-references.
- **-e** Turn on constant cross-references.
- **-f** Turn on function cross-references.
- **-l** Turn on label cross-references.
- **-m** Turn on manifest cross-references.
- **-v** Turn on variable cross-references.
- **-y** Turn on typedef cross-references.

-I*pathname*
Adds *pathname* to the list of directories to be searched when an **#include** file is not found in the directory containing the current source file or whenever angle brackets (< >) enclose the filename. If the file cannot be found in directories in this list, directories in a standard list are searched. The **-I** option can be used several times on the command line. This option only applies to source files appearing after it on the command line.

-L Turn off label cross-references.

-M Turn off manifest cross-references.

-N Enable cross-referencing of symbols in */usr/include*. By default these are not cross-referenced if they only appear in */usr/include* files.

-o*file*
Direct output to *file*, instead of standard output.

-s Operate silently; do not print input file names.

-t Format listing for 80-column width (default).

-U*name*, **-U***definition*
Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. For more information on the predefined symbols, see **cpp**(CP). Removes definition of *definition*, where *definition* is a manifest define. For more information on the manifest defines, see **cc**(CP).

-V Turn off variable cross-references.

-w<*num***>**
Width option which formats output no wider than <*num*> (decimal) columns. This option will default to 80 if <*num*> is not specified or is less than 51.

-Y Turn off typedef cross-references.

Files

LLIBDIR usually */usr/lib*

Diagnostics

Error messages are unusually cryptic, but usually mean that you cannot compile these files.

Notes

The **cxref** command considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program that **#includes** *ctype.h*, will contain many declarations of the variable **c**.

See also

cc(CP), **cpp**(CP)

Standards conformance

cxref is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

dbXtra

dbx-based Motif/X11 interface-oriented debugger

Syntax

dbXtra [*X11 flag* | *dbXtra flag*] ... [*objfile* [*coredump* | *processid*]]

Description

dbXtra is a debugger based on **dbx**, employing a **Motif** user interface. It is used for source-level debugging and program execution under SCO UNIX System V/386. *objfile* is an object file produced by **cc**(CP), using the **-g** flag to include symbol information. An object file created in this way contains a symbol table that includes information about the source code used to create the program. Only COFF files may be debugged using **dbXtra**.

If the file *.dbXtrarc* exists in the current directory or your home directory then the **dbXtra** commands in it are executed on start-up.

If a file called *core* exists in the current directory or a *coredump* file is specified on the command line, **dbXtra** can examine the state the program was in when it faulted.

If a *processid* is specified, **dbXtra** attempts to attach to the given process. This succeeds only if you are running on a version of SCO UNIX that supports the **ptrace**(S) **attach** request. If this is supported, processes can be attached to or detached from within **dbXtra**.

You may need to set the environment variable **DISPLAY** in order to run **dbXtra**. This variable informs your system of the address of the X display you are using.

Command line options

Any X11 command line option specified will be passed on to X11. The **dbXtra** command line options are:

-r Execute *objfile* immediately. If it terminates successfully, **dbXtra** exits. Otherwise the reason for termination is reported and you can either enter the debugger or let the program fault. **dbXtra** reads from */dev/tty* when **-r** is specified and standard input is not a terminal.

-x Ignore cross-reference file when initializing **dbXtra**.

-F **dbXtra** assumes that file-scoped structure/union/enum definitions with the same name are identical. Specifying **-F** may significantly decrease **dbXtra**'s memory usage (and startup time) for large programs. This happens because most structure definitions are in

#include files included in several source files; this causes many copies of the symbol table information to reside in the various corresponding *.o* files. Using this option means that the symbol table in the debugger may not match your program in some cases. The default is to assume that structure/union/enum definitions with the same name may have different definitions.

-I *dir* Add *dir* to the list of directories that are searched when looking for a source file. Normally **dbXtra** looks for source files in the current directory and in the directory where *objfile* is located. The directory search path can also be set with the **use** command.

-c*file* Execute the **dbXtra** commands in *file* before reading from standard input.

-C++ Assume C++ mode. **dbXtra** works in C++ mode if C++ information is present in *objfile*, and attempts to find extended debugging information files.

-C Specifically request C debugging only. C++ information is not used and toggling to C++ debugging mode is not allowed.

If neither **-C++** nor **-C** is specified, **dbXtra** looks for the function **_main()** in the file being debugged; if this function is found, **dbXtra** assumes C++ mode.

The Motif interface

The **Motif** interface to the **dbXtra** debugger conforms with the **Motif** style guide and has the "look and feel" of **Motif** applications. Refer to the *OSF/Motif Style Guide* for information on using the various graphical elements.

When **dbXtra** is invoked it creates one window, called the main window, with two panes within it. The pane at the top will be referred to as the source window, and the pane at the bottom will be referred to as the command window.

The main window

The main window is the principal window for user interaction during a debugging session. **dbXtra**'s main window is composed of a title bar, a menu bar on the top, command buttons on the left hand side, and the source window pane as well as the command window pane.

The title bar

The title bar displays the current debugging mode. This is [C++] for the C++ mode and [C] for the C mode. It also displays the current file and function names.

The source window pane

The source window shows the current position in the current file. This is a read-only view of the file—the program cannot be modified. The line numbers and breakpoint status are displayed to the left of the source window. The current execution point is indicated by an arrow [->]. Breakpoints are indicated by an [S] at the appropriate lines; trace events are indicated by a [T].

The command window pane

The command window provides direct access to the underlying **dbx** debugger. Output from most debugger commands will appear in this window and any **dbx** command can be entered using the keyboard from within this window.

Other windows

Other windows may appear on the screen from time to time as needed.

Program output window

The program output window appears when you are executing a program from **dbXtra** and displays the program's output. This is an **xterm** window and behaves like a terminal.

Information window

This kind of window displays the output of **dbXtra** commands that provide information to the user. These windows remain on the screen until dismissed by the user.

Display window

This window displays the values of selected symbols during program execution.

Pull-down menus

The menu bar is located across the top edge of the source window. The menus along the bar can be accessed by selecting the name of the menu using the mouse, or by using the keyboard accelerator sequence for the menu (simultaneously pressing the ⟨Alt⟩ key and the underlined, uppercase letter in the menu name). Each menu contains a group of buttons performing related operations.

The file menu

This menu has commands to save configuration information and exit **dbXtra**.

Write Configuration File
This command prompts for a filename, then creates a file containing **dbXtra** window geometry commands as X configuration commands suitable for inserting in a *.Xdefaults* or *.Xresources* file. See X(X) for more information.

Exit dbXtra
This command prompts to ensure the user really wants to quit, then exits **dbXtra** if so indicated.

The events menu

This menu contains a group of buttons which can be used to generate the various commands associated with breakpoint and trace events. The following is a brief description of the action performed by each button in the menu. The text between the "[" and the "]" characters following the name of each button is the **dbx** command constructed when the button is selected. For detailed information about the various **dbx** commands, see **dbxtra**(CP).

NOTE Commands in this menu which cause some action when a variable or address is accessed will have no effect when that access is performed by the kernel itself, for example, by a **read** statement.

Events [status]
This command displays the currently active events (breakpoints, trace expressions, etc.).

Call... [call <*function-call*>]
A dialog box appears, prompting the user for a function call. The function will then be invoked with the arguments specified. If the **$unsafecall** variable is not set, the parameters supplied for the call must match in number and type to the function. When **$unsafecall** is set the parameters do not have to match in number or type. If C syntax is used to call C++ member functions, all hidden parameters must be explicitly specified.

When... [when <*rest-of-command*>]
A dialog box appears, prompting the user for text which will be used in place of the <*rest-of-command*> field. The **when** command allows the user to execute one or more **dbx** commands whenever a specific event occurs during the execution of the user's program. For a complete explanation of the **when** command, see **dbxtra**(CP).

Stop -> This button generates a submenu which provides access to the various modes of the **stop** command.

if [stop if <*conditional-expression*>]
A dialog box appears, prompting the user for a conditional expression to further qualify the event. Execution of the debugged program will halt only if the conditional expression evaluates to "true".

function [stop in <function-name>]
A dialog box appears, prompting the user for a function name. The program will halt when the function is invoked.

address [stop inst at <address>]
A dialog box appears, prompting the user for an absolute memory address in the program space. The program will halt if this address is encountered in the execution stream.

line [stop at <line-number>]
A dialog box appears, prompting the user for a line number. The program will halt at the line specified.

access [stop access <variable> | <address>]
A dialog box appears, prompting the user for a variable or absolute memory address. The program will halt if the variable or address has been read from or written to.

NOTE Using this option requires the use of the debug registers.

change [stop change <variable> | <address>]
A dialog box appears, prompting the user for a variable or memory address name. The program will halt when that variable or address has been written to.

NOTE This option requires the use of the debug registers.

stop in main [stop in main]
This causes the program to halt before the first line in the function **main()** is executed. In C++ mode, all static initializers and constructors are executed before **main()** is called; use **stop in main** to regain control before static constructors are called.

stop [stop <*rest-of-command*>]
A dialog box appears, prompting the user for text which will be used in place of the <*rest-of-command*> field. This command allows specification of any variations of the **stop** command.

Trace -> This menu generates a submenu which provides access to the various modes of the **trace** commands.

function [trace <function-name>]
A dialog box appears, prompting the user for a function name. A message to be displayed when the function is invoked.

address [trace inst <address>]
A dialog box appears, prompting the user for an absolute address in the program space. A message will be displayed if that address is encountered in the execution stream

line [trace <line-number>]
A dialog box appears, prompting the user for a line number. A message will be displayed when that line is executed.

access [trace access <variable> | <address>]
A dialog box appears, prompting the user for a variable or absolute memory address. A message will be displayed. if the value at that address is read from or written to,

NOTE This command requires the use of the debug registers.

change [trace <variable>]
A dialog box appears, prompting the user for a variable name. A message is displayed when the variable has been written to. Writes of the same value will not halt the program.

NOTE This command requires the use of the debug registers.

trace [trace *<rest-of-command>*]
A dialog box appears, prompting the user for text which will be used in place of the *<rest-of-command>* field. This command allows specification of any variations of the **trace** command.

Interrupt
Sends SIGINT to the program. By default, the executing program is stopped and the source window is updated to display the current execution point.

Catch... [catch *<signal-name>*]
A dialog box appears, prompting the user to specify a signal. The signal may be specified by number or name (for example, SIGINT). The **catch** command adds the signal to the list of signals intercepted by the debugger. Upon receipt of the signal, the program is stopped. The user may then elect to send the signal to the program with the **continue** command. See the description of the **continue** command in **dbxtra**(CP).

Ignore... [ignore <*signal-name*>]
A dialog box appears, prompting the user to specify a signal. The signal may be specified by number or name (for example, SIGINT). The **ignore** command deletes this signal from the list of signals intercepted by the debugger. The signal will be processed by the program only.

Dbxtra
This command allows the user to enter complex commands for direct execution by **dbXtra**. The output from the debugger will be displayed in an information window, which can be removed when no longer required.

The execute menu

This menu is used to control the execution of the user's program.

Run... [run <*command-line-parameters*>]
A dialog box appears, prompting the user for the program command line parameters. The program is then run. Program execution will continue to completion unless a breakpoint or other event is encountered. A program output window is created allowing the user to interact with the program.

Rerun [rerun]
The **rerun** command restarts the program from its beginning. The previous command line parameters are used. Program execution will continue to completion unless a breakpoint or other event is encountered. Since **Rerun** does not prompt for parameters, the fastest way to run a program without parameters is to select this command. A program output window is created allowing the user to interact with the program.

Continue [cont]
The **continue** command continues execution from the current breakpoint or event.

Skip... [skip <*number-of-events*>]
A dialog box appears, prompting the user for the number of events to skip. Execution continues from the current location until the specified number of events have occurred.

Next [next]
The **Next** command executes one source line and then stops. If **Next** is used when the next line is a function call, **dbXtra** will cause the function to be executed but will not stop in the function's code. Use **Step** to follow execution into called functions.

Step [step]
The **Step** command executes one source line and then stops. **Step** will follow the execution into called functions.

Goto [goto <selected-line>]
goto inserts a temporary breakpoint at the selected source line, then continues execution of the program until it reaches that line.

Jump [jump <selected-line>]
Jump resumes execution of the program from the selected line.

Return [return]
Continue executing the program until it returns from the current function and stop in the calling function. The source window displays the line following the current function call.

Instruction ->
This submenu provides access to single-stepping by machine instructions.

The source menu

This menu is used to navigate the user's program source code.

Browse file... [file *<source-file-name>*]
A dialog box appears, prompting the user for a file name. The specified file is displayed in the source window.

Source Path... [use *<directory-list>*]
A dialog box appears, prompting the user for the list of directories used by **dbXtra** to find source files.

Search...
A dialog box appears, prompting the user for a regular expression search string. The pattern thus specified is then searched for within the displayed file. The search string defaults to the selected text in the source window.

Function... [func <function-name>]
A dialog box appears, prompting the user for a function name. The source text around the first executable statement within that function is then displayed.

Dbxref...
A dialog box appars, prompting the user for arguments to pass to **dbxref**. A cross-reference table is generated which is required for use of **xref**, **whatis**, and **where**. See **dbxtra**(CP) for more information on these commands.

Up [up]
The source window is changed to display the function that called the current function. This function now becomes the current function and variable scoping rules will default to this function.

Down [down]
The source window is changed to display the function that was called by the current function. This function now becomes the current function and variable scoping rules will default to this function.

The info menu

This menu is used to obtain information about the user's program.

Show... [whatis <*variable*>; print <*variable*>]
This command is used to show the type and value of a program variable or expression. If a variable or expression is selected in the source window, information about it is displayed; otherwise, a dialog box appears prompting the user for a variable or expression. The actual variable shown is determined by the scope rules of **dbx**. Explicit qualification can be used to avoid the default scoping rules. The optional keywords "enum", "union", or "struct" can be used to obtain information about aggregates. A function can be indicated with the "( )" characters appended to its name.

Where [where]
The **where** command displays the functions called by the user's program. The **up** and **down** commands can be used to change the current function to another one in the call stack.

Display... [display <*expr*>]
The **display** command is used to add a variable or expression to the display window. If a variable or expression is selected in the source window, it is added to the display window; otherwise, a dialog box appears prompting the user for a variable or expression. Information is updated and displayed whenever the debugger gains control. The display window is created if it does not exist. Items can be deleted by clicking on the item in the display window.

Assign... [assign <*stmt*>]
A dialog box appears, prompting the user for an assignment statement, which is then executed.

Which... [which <*name*>]
The **which** command is used to obtain a fully qualified name. The identifier is displayed in a fully qualified form with the names of the outer scopes prepended to the name. If a name is selected in the source window, information for that name is displayed; otherwise, a dialog box appears prompting the user for a name.

Print... [print <*expr*>]
The **print** command is used to display the value of an expression. If an expression is selected, its value is displayed;

otherwise, a dialog box appears prompting the user for an expression.

Xref... The **xref** command is used to print cross-reference information for a specified variable, function, or macro.

Variables [dump]
This command displays the names and values of variables local to the current function.

Registers [printregs]
This command displays the contents of the general and floating point registers of the processor.

Memory [<*address*>/*X*]
This command displays the contents of program memory.

The options menu

This menu is used to set dbXtra options and the environment variables of the program being debugged.

C/C++
The "C/C++" command allows the user to switch between the C++ source code and the C code that is a result of the compilation process. This is useful for programmers interested in the C language equivalent of a C++ construct. The C++ program must be compiled with the **-dbxC** flag.

Preferences ->
This submenu contains a number of **dbx** switches which can be toggled to suit user preferences. See the section on the **set** command in **dbxtra**(CP) for detailed information about these switches.

Environment ->
This submenu contains buttons to set and unset environment variables accessible by the program being debugged.

The help menu

This menu contains various buttons which provide on-line help information about the debugger.

The sidebar command buttons

Some frequently-used commands can be accessed via the "sidebar command buttons" on the left hand side of the main window. All these commands are available in the menus and have been described earlier. The commands are **Run...**, **Rerun**, **Stop**, **Continue**, **Next**, **Step**, **Interrupt**, **Goto**, **Jump**, and **Return**. The **Stop** command in this menu is the same as the **line** command under **Stop->** in the events menu.

The pop up menu

Associated with the source window is a "pop up" menu. This appears when the right mouse button is pushed. A command may be selected with the right mouse button. All commands in the pop up menu appear in another menu, and have been described earlier. Here is the list of commands: **Show...**, **Display...**, **Assign...**, **Which**, **Xref**, **Up**, and **Down**. The **Assign...** command in this menu allows a statement to be selected or entered in a dialog box.

Configurable resources

As with most X applications, you can configure a number of resources to modify the appearance of dbXtra to suit your needs. A number of the resources, for example to modify the font utilized, or the colors used, are described elsewhere. (See *OSF/Motif Style Guide* and *OSF/Motif User's Guide*, for example.) The values of the resources may be specified in the *.Xdefaults* file in your home directory, or in a file called *dbXtra* in */usr/lib/X11/app-defaults*, or in the *app-defaults* directory in your home directory.

The resources that control the appearance of **dbXtra** have the following format:

dbXtra***resource:value**

The following describes the various **dbXtra** resources that can be used:

configFile
The value for this resource is a string denoting the full pathname of the file containing the definitions for the various buttons and menus appearing in the source window. The default value for this is "*/usr/lib/dbxtra/dbXtra.config*".

lineNumberDigits
The value for this resource is an integer number which specifies the amount of space (in terms of digits) to be allocated for displaying line numbers in the source window. The default value for this resource is 4. A value of 0 will suppress the appearance of line numbers.

controlButtonWidth
The value for this resource is an integer number which specifies the amount of space (in terms of characters) to be used as the size of each button defined in the configuration file. This value also defines the number of characters that appear in the name of the button. If this value is not defined, it will be calculated to be the number of characters in the longest button-name in the configuration file.

controlPanel
The value for this resource controls the appearance of the control window. A value of "wide" means that the control window appears as a paned window underneath the source window. A value of "narrow" means that the control window appears as a paned window underneath the source window, and that the button column extends to the

bottom of the control window. The value of "separate" means that the control window will be completely separate from the source window. The default value for this resource is "wide".

font The value of this resource defines the font to be used to display characters. By default, this resource is not defined and therefore the default font used by the X session will be used.

control.font
The value of this resource defines the font to be used to display characters in the control window only. By default, this resource is not defined and therefore the default font used by the X session will be used.

display.font
The value of this resource defines the font to be used to display characters in the display window only. By default, this resource is not defined and therefore the default font used by the X session will be used.

output.font
The value of this resource defines the font to be used to display characters in the output window only. By default, this resource is not defined and therefore the default font used by the X session will be used.

control.wrap
The value of this resource is used to determine whether or not line wraparound is to be performed in the control window. If the value is "true", line wraparound will be performed. If the value is "false", wraparound will not take place and if any line exceeds the width of the window, a horizontal scroll bar will appear to allow access to the parts of the line which cannot fit within the available space.

help.wrap
The value of this resource is used to determine whether or not line wraparound is to be performed in the help window. If the value is "true", line wraparound will be performed. If the value is "false", wraparound will not take place and if any line exceeds the width of the window, a horizontal scroll bar will appear to allow access to the parts of the line which cannot fit within the available space.

information.wrap
The value of this resource is used to determine whether or not line wraparound is to be performed in the information windows. If the value is "true", line wraparound will be performed. If the value is "false", wraparound will not take place and if any line exceeds the width of the window, a horizontal scroll bar will appear to allow access to the parts of the line which cannot fit within the available space.

In addition to the above, various resources can be set to manipulate the various **Motif** widgets utilized. See *OSF/Motif User's Guide* for more details.

C++ debugging features

This version of the debugger supports C++ in a straightforward fashion; **dbXtra** is given knowledge of both C++ and C names and source files. At any time, the user may input either the C or C++ names for a function, structure member or variable; the **cc** command only affects the output from **dbXtra** to the user.

Classes in C++ become structures in C, reference variables (for example, **int&**) become pointers, and simple symbolic constants (*const*) become variables in the proper scope. Overloaded functions are prompted for. Enum types become ints, and enum constants become literal constants with no hint as to where they came from. Overloaded operator function names are of the form *operator+* (no blanks), where "+" is replaced by the overloaded operator. Overloaded new and delete operators are specified as *new* and *delete*, not *operatornew* and *operatordelete*.

To examine the translated C source along with the original C++ source, the source file must have been compiled with **CC -dbxC**; compiling with **-dbx** alone does not provide access to the C source.

Files

a.out	object file
tags	tags file
	source files

Comments

Since symbolic debugging information is replicated (both in the COFF file and in **dbXtra**) for each instance of a header file, storage is quickly taken up by replicated information when several source files **#include** the same header. In the case of C++ debugging, judicious use of the **-F** flag may save significant amounts of memory when starting up **dbXtra**.

Executable code within header files is not properly traced by **dbXtra**.

See also

cc(CP), **dbxtra**(CP), **ctags**(CP), **X**(X)

dbxtra

dbx-based screen-oriented debugger

Syntax

dbxtra [-r] [-s] [-x] [-F] [-I *dir*] [-c *file*] [-C] [-C++] [*objfile* [*coredump* | *processid*]]

Description

dbxtra is a screen-oriented debugger based on **dbx** for source level debugging and program execution under UNIX System V. *objfile* is a COFF-format object file produced by cc(CP) with the appropriate flag (**-g** or **-Zi**) specified to include symbol information. The machine-level facilities of **dbxtra** can be used on any program.

The object file contains a symbol table including the names of all source files the compiler used to create it. These files are available for perusal while using the debugger.

If a *core* file exists in the current directory or a *coredump* file is specified, **dbxtra** can be used to examine the state of the program when it faulted.

If a *processid* is specified, **dbxtra** attempts to attach to the given process. This only succeeds if UNIX supports the **ptrace**(S) **attach** request. If this is supported, processes can be attached to or detached from within **dbxtra**.

If the file *.dbxtrarc* exists in the current directory or your home directory then the **dbxtra** commands in it are executed on start-up.

Command line options

The **dbxtra** command line options are:

-r Execute *objfile* immediately. If it terminates successfully, **dbxtra** exits. Otherwise the reason for termination is reported and you can either enter the debugger or let the program fault. **dbxtra** reads from */dev/tty* when **-r** is specified and standard input is not a terminal.

-s Do not invoke the screen manager. **dbxtra** works in line mode only and will not respond to any of the screen-related commands (including **screen**).

-x Ignore cross-reference file when initializing **dbxtra**. (See **dbxref** command below.)

-F **dbxtra** assumes that file-scoped structure/union/enum definitions with the same name are identical. Specifying **-F** may significantly decrease **dbxtra**'s memory usage (and startup time) for large programs. This happens because most structure definitions are in include files, which produces many copies of the symbol table information in each *.o* file. Using this option means that the symbol table in the debugger may not match your program in some cases. The default is to assume that structure/union/enum definitions with the same name may have different definitions.

-I *dir* Add *dir* to the list of directories that are searched when looking for a source file. Normally **dbxtra** looks for source files in the current directory and in the directory where *objfile* is located. The directory search path can also be set with the **use** command.

-c *file* Execute the **dbxtra** commands in *file* before reading from standard input.

-C++ Assume C++ mode. **dbxtra** works in C++ mode if C++ information is present in *objfile*, and attempts to find extended debugging information files.

-C Specifically request C debugging only. C++ information is not used and toggling to C++ debugging mode is not allowed.

Unless **-r** is specified, **dbxtra** just prompts and waits for a command.

If neither **-C++** nor **-C** is specified, **dbxtra** looks for a function named **_main()** in the file being debugged, and, if found, assumes C++ mode.

Execution and tracing commands

run [*args*] [< *filename*] [>[> *filename*]]
rerun [*args*] [< *filename*] [>[> *filename*]]
Start executing *objfile*, passing *args* as command line arguments; " < ", " > ", or " >> " can be used to redirect input or output in the usual manner (" >> " appends to the file). Any arguments used with **rerun** are appended to the previous arguments and passed to the program; otherwise it is identical to **run**.

setenv *name value*
Set the value of environment variable *name* to be *value* in the environment of the debugged program. This command is useful whenever the debugged program requires a different value of environment variable than the value required by **dbxtra**. In particular, if the program's output is redirected to a terminal of a different type then the program requires a different value of the **TERM** variable.

unsetenv *name*
Unset the value of variable *name*, that is, remove it from the program's environment.

trace

The **trace** command is used to display source lines (or machine instructions) when executed, display the arguments and results of routines, and display when changes (or accesses) are made to variables or memory locations.

In the following, if an "**if** *condition*" clause is present, information is displayed only if *condition* is true. If an "**in** *routine*" clause is present, information is displayed only when *routine* is active. If *inst* is specified, **trace** displays machine instructions in addition to showing the source lines where they are executed.

trace [*inst*] [**in** *routine*] [**if** *condition*]
Display each source line before it is executed. If *inst* is specified, display each instruction, rather than each line.

trace [*inst*] *line-number* | *address* [**if** *condition*]
Display the given line-number before it is executed.

trace *routine* [**in** *routine2*] [**if** *condition*]
Display each call and return of *routine*.

The names of each caller is printed as well as the source line on which *routine* was called, and the parameters passed to it. In addition, *routine*'s return is noted, and if *routine* is a function then the return value is also printed.

trace [*inst*] [**change**] *variable* [**in** *routine*] [**if** *condition*]
When the *variable* is modified, display its new value before executing the next source line. This uses 386 debug registers, if possible.

trace [*inst*] **change** *address* [**in** *routine*] [**if** *condition*]
When the four bytes at location *address* are modified, display the new value before executing the next source line. This uses 386 debug registers, if possible.

trace [*inst*] **access** *variable* [**in** *routine*] [**if** *condition*]
When *variable* is modified or accessed, display its new value before executing the next source line. This must use 386 debug registers.

trace [*inst*] **access** *address* [**in** *routine*] [**if** *condition*]
When the four bytes at location *address* are modified or accessed, display the new value before executing the next source line. This must use 386 debug registers.

stop

The **stop** command is used to stop execution when execution reaches a given location, or when a variable or memory location is changed or accessed.

In the following, if "**if** *condition*" is present, stop only if *condition* is true. If *inst* is specified, stop before executing the next instruction, otherwise stop before the next source line is executed.

stop [*inst*] **if** *condition*
When the boolean *condition* becomes true, stop before the next source line or instruction in the program is executed. This uses 386 debug registers, if possible.

stop [*inst*] **at** *line-number* | *address* [**if** *condition*]
Stop at the given *line-number*. If *inst* is specified, stop at the given instruction *address*.

stop in *routine* [**if** *condition*]
Stop at the first executable statement in *routine*.

stop [*inst*] [*change*] *variable* [**if** *condition*]
Stop at the next source line after the *variable* is modified. This uses 386 debug registers, if possible.

stop [*inst*] [*change*] *address* [**if** *condition*]
Stop at the next source line after any of the the four bytes at memory location *address* are modified. This uses 386 debug registers, if possible.

stop [*inst*] **access** *variable* [**if** *condition*]
Stop at the next source line after the *variable* is modified or accessed. This must use 386 debug registers.

stop [*inst*] **access** *address* [**if** *condition*]
Stop at the next source line any of the after the four bytes at memory location *address* are modified or accessed. This must use 386 debug registers.

when

The **when** command is used to execute a series of *commands* when execution reaches a given location, when a condition is true, or when a variable or memory location is changed or accessed.

In the following, if "**if** *condition*" is present, perform the *commands* only if *condition* is true. If *inst* is specified, perform the *commands* before executing the next instruction, otherwise perform the *commands* before the next source line is executed.

when [*inst*] **at** *line-number* | *address* [**if** *condition*] {*command*; [*command*; ...]}
Execute the *commands* when execution reaches the given *line-number*. If *inst* is specified, execute the *commands* when execution reaches the given instruction *address*.

when in *routine* [**if** *condition*] {*command*; [*command*; ...]}
Execute the *commands* at the first executable statement of the routine.

when [*inst*] *condition* {*command*; [*command*; ...]}
When the boolean *condition* is true, execute the *commands* before each source line is executed. This uses 386 debug registers, if possible.

when [*inst*] *change variable* [**if** *condition*] {*command*; [*command*; ...]}
When *variable* is modified, execute the *commands* before the next source line is executed. This uses 386 debug registers, if possible.

when [*inst*] *change address* [**if** *condition*] {*command*; [*command*; ...]}
When any of the four bytes at memory location *address* are modified, execute the *commands* before the next source line is executed. This uses 386 debug registers, if possible.

when [*inst*] **access** *variable* [**if** *condition*] {*command*; [*command*; ...]}
When the *variable* is modified or accessed, execute the *commands* before the next source line is executed. This must use 386 debug registers.

when [*inst*] **access** *address* [**if** *condition*] {*command*; [*command*; ...]}
When any of the the four bytes at memory location *address* are modified or accessed, execute the *commands* before the next source line is executed. This must use 386 debug registers.

Other execution and tracing commands

status [> *filename*]
Print out the currently active **trace**, **when** and **stop** commands. Inactive events (which have been made inactive with the **disable** command) are shown with an asterisk (*) after the *event-number*.

delete *event-number* ...
The **trace**, **stop**, or **when** commands corresponding to the given *event-number* are removed. The *event-number* associated with **traces**, **stops** and **whens** are printed by the **status** command.

disable *event-number* ...
The **trace**, **stop**, or **when** commands corresponding to the given *event-number* are disabled. Disabled events do not trigger until they are **enabled**.

enable *event-number* ...
The **trace**, **stop**, or **when** commands corresponding to the given *event-number* are enabled.

catch *number*
catch *signal-name*
ignore *number*
ignore *signal-name*
Start or stop trapping a signal before it is sent to the program. This is useful when a program being debugged handles signals such as interrupts. A signal may be specified by number or by a name (for example, **SIGINT**). Signal names are case insensitive and the **SIG** prefix is optional. By default all signals are trapped except **SIGCONT**, **SIGCHILD**, **SIGALRM** and **SIGKILL**.

cont *integer*
cont *signal-name*
Continue execution from where it stopped. If *signal-name* is specified, the process continues as though it received the signal. Otherwise, the process is continued as though it had not been stopped.

Execution cannot be continued if the process has "finished", that is, called the standard procedure **exit**. **dbxtra** does not allow the process to exit, thereby letting you examine the program state.

skip [*n*]
Continue execution from where it stopped. If *n* is specified, that many breakpoints are ignored before the program stops. If *n* is not given, one breakpoint is skipped. Execution cannot be continued if the process has "finished", that is, it has called the standard procedure **exit**.

step [*n*]
Execute *n* source lines (default value 1).

next [*n*]
Execute up to the next *n* source lines (default value 1). The difference between this and **step** is that if the line contains a call to a procedure or function the **step** command stops at the beginning of that block, while the **next** command does not.

return [*procedure*]
Continue until a return to *procedure* is executed, or until the current *procedure* returns if none is specified.

call *procedure(parameters)*
Execute the object code associated with the named *procedure* or *function*.

jump *line-number*
Continue execution from the specified *line-number*. The *line-number* must be in the current function.

goto *line-number*
Continue execution and stop before the first instruction at the specified *line-number* is executed. This creates in effect a temporary break point which only affects execution when the command is used. A *filename*:*line-number* pair with *filename* in double quotes can be used to specify a *line-number* outside the current file.

Screen related commands

The commands in this section are effective only when **dbxtra** is invoked in screen mode (the default mode). In this mode the screen is divided into two equal sized windows. The top window contains program source and the bottom window is used for the usual **dbxtra** session (that is, **dbxtra** commands and output and the program's input and output). The separating bar (in reverse video) shows the current file and procedure name. A pointing arrow "->" on the left marks the current execution line (the line to be executed next).

NOTE In an emergency situation it is undesirable to kill **dbxtra** with SIGKILL because it leaves the screen in an unknown mode, and may also leave some orphaned processes. Instead, use three SIGQUIT signals in a row without any input between them.

screen [on | off]
Turn screen mode on/off (no argument is equivalent to on). If screen mode is already on, **screen** redraws the screen.

more [on | off]
Turn on/off **more**-like features for **dbxtra** and the debugged program's output (no argument is equivalent to on). The line count is reset each time the keyboard is touched so that only lines between commands are **more**'d.

vt100 [on | off]
Turn on/off **vt100** emulator for the debugged programs output, (default is on). The vt100 emulation facility allows full screen applications to be debugged with **dbxtra**. The debugger provides an emulated vt100 screen for program input/output. This screen is displayed automatically when the debugged program is running.

forward [*n*]
backward [*n*]
Scroll the source window forward and backward *n* lines (the default is one window).

display [[*line-number*] *expression* [, *expression* ...]]
undisplay [*expression-number* | *expression* ...]

display puts the value of its argument list of expressions in a special-purpose window. The values are updated whenever the program stops execution. The named *expressions* are added to the currently displayed expressions on a new line in the display window (**display** *a,b* puts *a* and *b* on the same line, whereas **display** *a* ; **display** *b* puts them on two separate lines). If the optional *line-number* is specified, the expressions are added to the display line given by the number. If no arguments are given, the list of displayed expressions is printed, each preceded by an index number that can be used as an argument for **undisplay**.

undisplay removes one or more expressions from the display window. If *expression-number* is specified, the display line corresponding to that number is removed from the display. If an *expression* is specified, that *expression* is removed from within one of the display lines. If no arguments are given to **undisplay**, then the last line of expressions is removed.

Function keys

dbxtra allows the user to associate a function key or a sequence of keys with one or more screen functions. The following commands are used to define and print keys and turn key interpretation on and off.

bindkey [*key* [*string*]]
unbindkey *key*

Bind (unbind) *key* with the contents of *string*.

key can be specified as a *terminfo* key name (for example, key_up, key_f1) or as a string of characters. Control characters that are part of the key can be typed verbatim, or specified in ^*X* notation (for example, ⟨Esc⟩ is ^[) or specified in the *nnn* notation (where *nnn* is the octal representation of the ASCII value).

string contains one or more screen editing commands separated by a semi-colon (" ; ").

If *string* is not specified to the **bindkey** command, **dbxtra** prints the definition of the given *key*. If *key* is not specified **dbxtra** prints all key definitions. The definitions are printed according to the **bindkey** command syntax such that if the output of **bindkey** is redirected to a file, that file can be used to create key definitions in subsequent sessions.

keys [**on** | **off**]

Turn on/off interpretations of keys into screen functions (if no argument is provided, it defaults to on). Turning off screen mode (**screen off**) automatically turns off key interpretation.

Screen functions

Functions with arguments take the string following the function name to the next delimiter (" ; ") or to the end of string. If the argument contains a " ; " it must be preceded by a backslash (" \ "). Variables (*name*) in the argument are evaluated before being passed to the function.

The following screen functions are implemented:

bof	Cursor to end of file.
bol	Cursor to beginning of line.
close	Close a window.
cursor_down	One line down.
cursor_left	One character left.
cursor_right	One character right.
cursor_up	One line up.
del_bol	Delete from cursor location to beginning of line (only on entered line).
del_char	Delete character at cursor location (only on entered line).
del_word	Delete a word to the left of the cursor (only on entered line).
eof	Cursor to end of file.
eol	Cursor to end of line.
expand	Expand current window by two lines.
expand_lower	Expand current window at lower edge by one line.
expand_upper	Expand current window at upper edge by one line.
goto_line	Cursor to line number.
goto_mark	Cursor to mark set by **set_mark** (see below).
half_page_back	Scroll half page backward.
half_page_forw	Scroll half page forward.
load_file	Load file into current window.
line_back	Scroll one line backward.
line_forw	Scroll one line forward.
move_capability	Move a named capability to current window.
next_command	Copy next **dbxtra** command as current command.
next_window	Move cursor to next window.

page_back	Scroll one page backward.
page_forw	Scroll one page forward.
prev_command	Copy previous **dbxtra** command as current command.
prev_window	Move cursor to previous window.
quote_char	Accept next char as is (escape key interpretation).
redraw	Redraw the screen.
screen_function	Enter a screen function by name rather than key.
send_echo	Send the argument to **dbxtra** with echo.
send_edit	Same as above but allow line editing before sending.
send_noecho	Send the argument to **dbxtra** with no echo.
set_mark	Set a mark in a window for use by **goto_mark**.
show_xref	Show the next source location given by **xref**. **xref** displays a list of line numbers for lines containing the specified symbol. To put such a line at the top of source window, move the cursor in the bottom window to the line number in question, and press the character sequence bound to **show_xref**.
shrink	Shrink current window by two lines.
split	Split a window into two. Splitting **dbxtra** session window creates a program window.
toggle_follow_cursor	Toggles between the static portion of the full screen window and the portion of the window containing the cursor. This is only useful when the program window is being used.
toggle_full_screen	Toggles between the full screen window and the standard dbx/source window layout. Since this is a screen function, it can only be used when the program is not executing.
toggle_line_numbers	Toggle with/without line numbers.
toggle_more	Toggle more/nomore for current window.
toggle_wrap_around	Toggle wrap around of long lines.
word_left	One word left.
word_right	One word right.

The following screen function variables are implemented and evaluate to:

$command	**dbxtra** command typed by the user at cursor line.
$events	Event ids of breakpoints on cursor line.
$expr	Like *$var* but including matching parentheses, field operators (" . " and " -> ") and spaces following dereference operator (" * . ")

$file	Source file name in current window.
$keyboard	String entered at the keyboard at evaluation time.
$line	Source line number of cursor position.
$var	The longest string of alphanumeric and underscore ("_") characters at the cursor position.

Printing variables and expressions

Names are resolved first using the static scope of the current function, then using the dynamic scope if the name is not found in the static scope. If static and dynamic searches do not yield a unique symbol, an arbitrary symbol is chosen and the following message is printed:

```
[using qualified name]
```

The name resolution procedure may be overridden by qualifying an identifier with a block name, for example, ***module.variable***. For C and C++, source files are treated as modules named by the file name without *.c*.

dbxtra attempts to choose the most reasonable symbol for a given context. For example, a **call** ***symbol(exp)*** only looks for a ***symbol*** that is a function or procedure. If this is not the desired behavior, then the command **set $allmatch** allows any symbol to match in any context. This context checking is most useful for C++ debugging.

Expressions are specified with an approximately common subset of C and Pascal syntax. Indirection can be denoted using either a prefix "*" or a postfix "^" and array expressions are subscripted by brackets ("[ ]"). Array subranges (slices) are separated by two dots ("..") and are subscripted by brackets ([***exp..exp***]). The field reference operator (".") can be used with pointers as well as records, making the C and C++ operator "->" unnecessary (although it is supported).

Types of expressions are checked; the type of an expression may be overridden by using ***type-name(expression)***. An alternate form of overriding types is ***expression \ typename***. A pointer to a named type may be constructed with ***expression \ & typename***.

A limited form of C casts are supported using (***type***) ***expression***. ***type*** may be any of the C basic types (such as ***int*** or ***char***) or a C structure type, followed by any number of asterisks ("*.") Due to an implementation restriction, typedef names may not be used.

C structure types are represented by **struct** ***name***. In the same manner, **union** ***name***, **enum** ***name*** and **class** ***name*** may be used. Using this form, ***name*** is a single unqualified word.

assign ***variable = expression***
Assign the value of ***expression*** to ***variable***.

dump [*procedure*] [> *filename*]
Print the names and values of variables in *procedure*, or the current one if none is specified. If the *procedure* given is " . ", then all active variables are dumped.

print *expression* [, *expression* ...]
Print out the values of the *expressions*.

whatis *expression*
Print the declaration of the given *expression*. If cross-reference information has been loaded, macro definitions are printed.

which *identifier*
Print the full qualification of the given *identifier*, that is, the outer blocks that the identifier is associated with.

up [*count*]
down [*count*]
Move the current function, which is used for resolving names, up or down the stack *count* levels. The default *count* is 1.

where
Print out a list of the active procedures and function.

whereis *identifier*
Print the full qualification of all the symbols whose name matches the given *identifier*. If cross-reference information has been loaded, print the file names and line numbers where each global symbol is defined. The order in which the symbols are printed is not meaningful.

dbxref [-i] [-o*file*] [*options*] [*files*]
Generate the cross-reference file (default name *dbxtra.xref*) necessary for the **xref** and enhanced **whatis**, and **whereis** commands. *options* are the **cc**(CP) options **-I**, **-D**, and -U, and *files* is the list of C files. The **-i** option tells **dbxref** to do an incremental cross-reference. If *files* is omitted, all C files that are referenced in the object file are used. The **-o***file* option tells **dbxref** to place the output in *file*. An extension of *.xref* is added if necessary. The command to invoke **dbxref** is passed to the user's shell, so all shell environment variables and meta-characters may be used.

An incremental cross-reference is used to update the cross-reference when only some of many files have been modified. The old information relevant to those files is forgotten, and the files are re-scanned, and the information integrated with the old data.

xref *identifier*
If cross-reference information has been loaded, print a cross-reference for the given symbol. Only global variables, functions, and macros may be **xref**ed. The cross-reference information is generated with the **dbxref** command. If the requested name is a function then the following information is printed:

Referenced from:
A list of functions which call the named function. Each function name is followed by the filename in which the call occurs, and one or more line numbers.

Reference to functions/macros:
A list of functions and macros with arguments called by the named function. Each function name is followed by one or more line numbers where the call occurs. An **M** is present at the beginning of the line if this is a macro call.

Reference to variables/macros:
A list of global variables and parameterless macros referenced by the named function. Each variable name is followed by one or more line numbers where the reference occurs, but there is no distinction between the use of a variable and an assignment to a variable. An **M** is present at the beginning of the line if this is a macro reference.

If the name given to **xref** is a macro or a global variable, then only the *referenced from* information is given. The **xref** command does not accept other argument types (local variables, type names or structure tags).

If C++ mode is on, **xref** displays the classes that a subroutine or class is a friend to.

Accessing source files

/*regular expression*[/]
?*regular expression*[?]
Search forward or backward in the current source file for the given pattern.

edit [*filename*]
edit *procedure/function-name*
Invoke an editor on *filename* or the current source file if none is specified. If a *procedure* or *function* name is specified, the editor is invoked on the file that contains it. Which editor is invoked by default depends on the installation. The default can be overridden by setting the environment variable **EDITOR** to the name of the desired editor.

file [*filename*]
Change the current source file name to *filename*. If none is specified then the current source file name is printed.

func [*procedure/function*]
Change the current function. If none is specified then print the current function. Changing the current function implicitly changes the current source file to the one that contains the function; it also changes the current scope used for name resolution.

list .
list *procedure/function*
list [*source-line-number* [, *source-line-number*]]
List the lines in the current source file from the first *line-number* to the second inclusive. If no lines are specified and the user is in screen mode, then no new source lines are displayed and the user is informed about the **forward** command. Otherwise, the next 10 lines are listed. If the name of a procedure or function is given lines *n-k* to *n+k* are listed where *n* is the first statement in the procedure or function and *k* is a small number that depends on the size of the window. If the argument is " . " then the current execution location is listed. (Note that **list .** only changes the source location and not the current function for scope purposes).

use *directory-list*
Set the list of directories to be searched when looking for source files.

Command aliases

alias
alias *name name*
alias *name "string"*
alias *name* (*parameters*) *"string"*
When commands are processed, **dbxtra** first checks to see if the word is an alias for either a command or a string. If it is an alias, then **dbxtra** treats the input as though the corresponding string (with values substituted for any parameters) had been entered. For example, to define the alias **rr** for the command **rerun**, type:

```
alias rr rerun
```

To define an alias called **b** that sets a stop at a particular line, type:

```
alias b(x) "stop at x"
```

Subsequently, the command **b(12)** expands to **stop at 12**.

If no parameters are specified, the alias command prints all the currently defined aliases (most **dbxtra** commands have predefined aliases).

Aliases are not transitive. You cannot do **alias x run; alias y x**, and expect *y* to execute the **run**.

unalias *name*
Remove the alias with the given *name*.

set *name* [= *expression*]
The **set** command defines values for debugger variables. The name of a debugger variable cannot conflict with the names of variables in the program being debugged. When used within other commands, a debugger variable is expanded to its corresponding expression.

unset *name*
Delete the debugger variable associated with *name*.

Debugger variables

The following debugger variables have a special meaning:

$hexchars
$hexints
$hexoffsets
$hexstrings
When set, **dbxtra** prints out characters, integers, offsets from registers, and character pointers respectively in hexadecimal.

$listwindow
The value of this variable specifies the number of lines to list around a function when the **list** command is given a function or procedure as its argument. The default value is 10.

$printbases
When set, members of base classes inherited by the current object or class are printed in **print** or **whatis** command output.

$unsafecall
$unsafeassign
When **$unsafecall** is set, strict type checking is turned off for arguments to subroutine or function calls (for example, in the **call** statement). When **$unsafeassign** is set, strict type checking between the two sides of an **assign** statement is turned off. These variables should be used only with great care, because they severely limit **dbxtra**'s usefulness for detecting errors.

$sleep
Specify the number of seconds that **dbxtra** sleeps at each statement when tracing the whole program in single-step (in screen mode only). The default value is 1 second.

$nodebugregs
When **$nodebugregs** is set, **dbxtra** does not attempt to use 386 hardware debug registers. The **trace access**, **when access**, and **stop access** commands need the registers enabled to function. The hardware debug registers will not detect access or modification performed by UNIX (such as **read** and **write** system calls), and it may be desirable to turn off use of the debug registers.

$nodebugwarn
When **dbxtra** attempts to use 386 debug registers, and there are no free registers, a warning message is printed, and slow emulation is performed. If **$nodebugwarn** is set, the warning message is suppressed.

$allmatch
When **$allmatch** is set, the choosing of the *correct* symbol in a given context is not done. In this case, all symbols are equally *reasonable*.

$noclobber
If set then redirecting the output of a command into an existing file fails.

$windowshell
If set then the **sh** command invokes a shell in the interaction window instead of turning the screen mode off automatically.

$stringlen
Gives the number of characters (rounded up to the next word boundary) to be printed from a character pointer without seeing a null. If a string is truncated by reaching the **$stringlen** limit, it is terminated with an ellipsis. **$stringlen** defaults to 512.

$machinecode
Setting **$machinecode** affects the disassembly output. The hexadecimal contents of the instruction locations are shown in place of the *function:line-number* field.

$pccformat
Setting **$pccformat** causes the disassembly instruction format to be the format generated by the AT&T C compiler (**rcc**). Under SCO UNIX, the default disassembly instruction format is the format generated by the Microsoft C compiler **cc**.

$fork
If UNIX supports the **ptrace**(S) **stop on fork** request, **dbxtra** allows debugging of children of the debugged process. If **$fork** is not set, the child process is ignored, and executes to completion.

$fork may be set to a string with the following values:

parent	**dbxtra** continues debugging the parent process. The child process is run to completion.
child	**dbxtra** begins debugging the child process. The parent process is run to completion.
push	**dbxtra** pushes a new **dbxtra** to debug the child process. **dbxtra** resumes debugging the parent process when the child **dbxtra** exits.
both	If **dbxtra** is running under the Motif screen manager, a new Motif **dbxtra** will be started to debug the child process.
prompt	The user is asked to choose one of the above choices.

$noexec
If UNIX supports the **ptrace**(S) **stop on fork** request, **dbxtra** determines which file has been **exec**'ed, and switches to the correct file. If **$noexec** is set, **dbxtra** does *not* switch executable files. The **exec** command may be used to manually change files.

Machine level commands

stepi
nexti
Single-step as in **step** or **next**, but do a single instruction rather than source line.

regs
Display the contents of the 80386 registers in a human-readable form.

fpregs
Display the contents of the 80387 floating point registers in a human-readable form. The contents of the control and status words are displayed, and then the floating point register stack is shown.

address, address/ [mode]
address / [count] [mode]
Print the contents of memory starting at the first ***address*** and continuing up to the second ***address*** or until ***count*** items are printed. If the address is " . ", the address following the one printed most recently is used. The ***mode*** specifies how memory is to be printed; if it is omitted the previous mode specified is used. The initial mode is X. The following modes are supported:

i	print the machine instruction
d	print a short word in decimal
D	print a long word in decimal
u	print a short word in unsigned decimal
U	print a long word in unsigned decimal
o	print a short word in octal
O	print a long word in octal
x	print a short word in hexadecimal
X	print a long word in hexadecimal
b	print a byte in octal
c	print a byte as a character
s	print a string of characters terminated by a null byte
f	print a single precision real number
g	print a double precision real number

Symbolic addresses are specified by preceding the name with an **&**. Addresses may be expressions made up of other addresses and the operators " + ", " - ", " * ", unary " - ", unary " * " (dereference), each operator assuming its usual precedence. This precedence can be overridden using parentheses " (" and ") ".

Miscellaneous commands

cc Toggle between C++ and C output modes. This command is not enabled if the **-C** (C code only) flag was specified when invoking **dbxtra**. When in C++ output mode, all names and source files output by **dbxtra** refer to C++ names and source; when in C mode, all names output are C names, and thus somewhat confusing when debugging a C++ program. Both C++ and C names may be specified by the user in either mode. To examine the C source produced by the C++ compiler, the program must be compiled with **CC -dbxC** instead of **CC -g** or **CC -dbx**.

help [*command*]
help [*string*]
Print out a synopsis of **dbxtra** commands or subjects. If *command* is specified, the section from the manual page for that command is printed. For commands (or subjects) that include non-alphanumeric characters the command name must be quoted (for example, **help "/"**).

quit Exit **dbxtra**.

sh [*command-line*]
! *command-line*
Pass the *command-line* to the shell for execution. If no command is specified then invoke an interactive shell. The **SHELL** environment variable determines which shell is used.

source *filename*
Read **dbxtra** commands from the given *filename*.

kill This kills the current process being debugged. After a *kill*, the executable file is not active, and it may be recompiled. At the next **run** or **rerun** command, **dbxtra** notices the file has changed, and rereads the symbol table.

detach
If the version of UNIX supports the **ptrace**(S) **detach** request, the process being debugged has all breakpoints removed, and it is allowed to execute at full speed. If **detach** is not supported, the process is killed.

exec *filename*
If the process being debugged does an **exec**(S) system call, this tell **dbxtra** that a new symbol table should be read in.

version
Print the version information for dbXtra. This should be given for support calls.

C++ debugging features

This version of the debugger supports C++ in a straightforward fashion; **dbxtra** is given knowledge of both C++ and C names and source files. At any time, the user may input either the C or C++ names for a function, structure member or variable; the **cc** command only affects the output from **dbxtra** to the user.

Classes in C++ become structures in C, reference variables (for example, **int&**) become pointers, and simple symbolic constants (*const*) become variables in the proper scope. Overloaded functions are prompted for. Enum types become ints, and enum constants become literal constants with no hint as to where they came from. Overloaded operator function names are of the form *operator+* (no blanks), where "+" is replaced by the overloaded operator. Overloaded new and delete operators are specified as *new* and *delete*, not *operatornew* and *operatordelete*. To examine the translated C source along with the original C++ source, the source file must have been compiled with **CC -dbxC**; compiling with **-dbx** alone does not provide access to the C source.

Files

a.out	object file
dbxtra.xref	cross-reference file

Comments

dbxtra is intended for use on COFF object files only. **dbxtra** will generate a `bad magic number` error message if it is invoked on non-COFF object files.

Since symbolic debugging information is replicated (both in the COFF file and in **dbxtra**) for each instance of a header file, storage is quickly taken up by replicated information when several source files **#include** the same header. In the case of C++ debugging, judicious use of the **-F** flag may save significant amounts of memory when starting up **dbxtra**.

Executable code within header files are not properly traced by **dbxtra**.

See also

cc(CP)

delta

make a delta (change) to an SCCS file

Syntax

delta [-r*SID*] [-s] [-n] [-g*list*] [-m[*mrlist*]] [-y[*comment*]] [-p] *files*
delta -

Description

The **delta** command is used to introduce into the named SCCS file changes that were made to the file retrieved by **get**(CP) (called the *g-file*, or generated file).

The **delta** command makes a delta to each named SCCS file. If a directory is named, **delta** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read (see "Warnings"); each line of the standard input is taken to be the name of an SCCS file to be processed.

The **delta** command may issue prompts on the standard output depending upon certain options specified and flags (see **admin**(CP)) that may be present in the SCCS file (see **-m** and **-y** options below).

Options apply independently to each named file.

-r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this option is necessary only if two or more outstanding **gets** for editing (**get -e**) on the same SCCS file were done by the same person (login name). The *SID* value specified with the **-r** option can be either the *SID* specified on the **get** command line or the *SID* to be made as reported by the **get** command (see **get**(CP)). A diagnostic results if the specified *SID* is ambiguous or if a required *SID* is omitted on the command line.

-s Suppresses the issue, on the standard output, of the created delta's *SID*, as well as the number of lines inserted, deleted, and unchanged in the SCCS file.

-n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

-g*list* a list (see **get**(CP) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (*SID*) created by this delta.

-m[*mrlist*]
If the SCCS file has the **v** flag set (see **admin**(CP)), then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

If **-m** is not used and the standard input is a terminal, the prompt `MRs?` is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The `MRs?` prompt always precedes the `comments?` prompt (see **-y** option).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the **v** flag has a value (see **admin**(CP)), it is taken to be the name of a program (or shell procedure) which validates MR numbers. If a non-zero exit status is returned from the MR number validation program, **delta** terminates. (It is assumed that the MR numbers were not all valid.)

-y[*comment*]
Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If **-y** is not specified and the standard input is a terminal, the prompt `comments?` is issued on standard output before standard input is read; if standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-p Causes **delta** to print (on standard output) the SCCS file differences before and after the delta is applied in a **diff**(C) format.

Files

d-file	Created during the execution of **delta**; removed after completion.
g-file	Existed before the execution of **delta**; removed after completion.
p-file	Existed before the execution of **delta**; may exist after completion.
q-file	Created during the execution of **delta**; removed after completion.
x-file	Created during the execution of **delta**; renamed to SCCS file after completion.
z-file	Created and removed during the execution of **delta**.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

Warnings

Lines beginning with an ASCII SOH character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile*(FP)) and will cause an error.

A **get** of many SCCS files, followed by a **delta** of those files, should be avoided when the **get** generates a large amount of data. Instead, multiple **get/delta** sequences should be used.

If standard input ("-") is specified on the **delta** command line, the **-m** (if necessary) and **-y** options *must* also be present. Omission of these options causes an error.

Comments are limited to text strings of at most 512 characters.

See also

admin(CP), **bdiff**(C) **cdc**(CP), **get**(CP), **prs**(CP), **rmdel**(CP), **sccsfile**(FP)

Standards conformance

delta is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

dis

object code disassembler

Syntax

dis [**-d** *sec*] [**-da** *sec*] [**-F** *function*] [**-L**] [**-l** *string*] [**-o**] [**-R**] [**-s**] [**-T**] [**-t** *sec*] [**-V**] *file* ...

Description

The **dis** command produces an assembly language listing of *file*, which may be an object file or an archive of object files. The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements. This utility operates only on COFF executables.

dis accepts the following options which may be specified in any order.

-d *sec* — Disassemble the named section as data, printing the offset of the data from the beginning of the section.

-da *sec* — Disassemble the named section as data, printing the actual address of the data.

-F *function* — Disassemble only the named function in each object file specified on the command line. The **-F** option may be specified multiple times on the command line.

-L — Look up source labels in the symbol table for subsequent printing. This option works only if the file was compiled with additional debugging information (for example, the **-g** option of **cc**(CP)).

-l *string* — Disassemble the library file specified by *string*. For example, one would issue the command **dis -l x -l z** to disassemble *libx.a* and *libz.a*. All libraries are assumed to be in *LIBDIR*.

-o — Print numbers in octal (the default is hexadecimal).

-R — Real mode flag. **dis** assumes 16-bit wide addresses and registers when disassembling a file using the **-R** option.

-s — Perform symbolic disassembly, that is, specify source symbol names for operands where possible. Symbolic disassembly output will appear on the line following the instruction. For maximal symbolic disassembly to be performed, the file must be compiled with additional debugging information (for example, the **-g** option of **cc**(CP)). Symbol names will be printed using C syntax.

-T Debug the operation of the disassembler itself.

-t *sec* Disassemble the named section as text.

-V Print, on standard error, the version number of the disassembler being executed.

If the **-d**, **-da** or **-t** options are specified, only those named sections from each user-supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as **[5]**, indicates that the break-pointable line number starts with the following instruction. These line numbers will be printed only if the file was compiled with additional debugging information (for example, the **-g** option of **cc**(CP)). An expression such as **<40>** in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A function name will appear in the first column, followed by **( )**.

Files

LIBDIR Usually */lib*.

Diagnostics

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

See also

a.out(FP), **as**(CP), **cc**(CP), **ld**(CP)

Standards conformance

dis is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

dosld

MS-DOS cross linker

Syntax

dosld [*options*] *file* ...

Description

dosld links the object *file*(s) to create a program for execution under MS-DOS. Although similar to **ld**(CP), some **dosld** options differ significantly from **ld**. The options are described below:

-C Case sensitivity on. This option instructs **dosld** to treat upper and lower case characters in symbol names as distinct (so that the character " A " is different from the character " a ").

-D DS Allocate. This instructs **dosld** to perform DS allocation. It is generally used in conjunction with the **-H** option.

-F *num*
Set stack size to the hexadecimal value *num*. **dosld** will set the stack segment to *num* bytes in the output file.

-G Ignore group associations. This option instructs **dosld** to ignore any group definitions it may find in the input files. This option is provided for compatibility with old versions of MS-LINK; generally, it should not be used.

-H Load high. This option instructs **dosld** to set a field in the header of the executable file to tell MS-DOS to load the program at the highest available position in memory. It is most often used with programs in which data precedes code in the memory image.

-L Include line numbers. This option instructs **dosld** to include line numbers in the listing file (if any). **dosld** cannot put line numbers in the listing file if the source translator hasn't put them in the object file.

-M Include public symbols. This option instructs **dosld** to include public symbols in the list file. The symbols are sorted twice: lexicographically and by address.

-m *filename*
Create a file with the given name including information about the segments and groups in the executable *filename*. Public symbols and line numbers will be listed in this file if the **-L** and **-M** options are given.

-nl *num*
Set name length to a decimal number, *num*. **dosld** will truncate all public and external symbols longer than *num* characters.

-o *filename*
Create output file *filename*, instead of the default name, *a.out*.

-S *num*
Set segment limit to a decimal number, *num*, between 1 and 1024 (the default value is 128). The number sets the limit on the number of different segments that may be linked together. The higher the value you set, the slower the link will be.

-u *name*
Name undefined symbol, *name*. **dosld** will add *name* to its symbol table as an undefined symbol. The **-u** option may appear several times on the command line.

As with **ld**, the files passed to **dosld** may be either UNIX-style libraries (objects collected using **ar**(CP)) or ordinary 8086 object files. Unless the **-u** option appears, at least one of the files passed to **dosld** must be an ordinary object file. Libraries are searched only after all the ordinary object files have been processed.

Files

/usr/bin/dosld

See also

ar(CP), **as**(CP), **cc**(CP), **ld**(CP), **ranlib**(XNX)

Standards conformance

dosld is a value-added extension of AT&T System V provided by the Santa Cruz Operation.

dump

dump selected parts of a common object file

Syntax

dump [*options*] *files*

Description

The **dump** command outputs selected parts of each of its object *files*. The output can be directed via the standard redirection tools. **dump** operates only on COFF executables.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

-a Dump the archive header of each member of each archive file argument.

-c Dump the string table.

-f Dump each file header.

-g Dump the global symbols in the symbol table of an archive.

-h Dump section headers.

-L Interpret and print the contents of the *.lib* sections.

-l Dump line number information.

-o Dump each optional header.

-r Dump relocation information.

-s Dump section contents.

-t Dump symbol table entries.

-V Print the **dump** version number.

-z *name* Dump line number entries for the named function.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.

-d *number* Dump the section number, *number*, or the range of sections starting at *number* and ending at the *number* specified by **+d**.

+d *number* Dump sections in the range either beginning with first section or beginning with section specified by **-d**.

-n *name* Dump information pertaining only to the named entity. This *modifier* applies to **-h**, **-l**, **-r**, **-s**, and **-t**.

-p Suppress printing the headers.

-t *index* Dump only the indexed symbol table entry. The **-t** used in conjunction with **+t**, specifies a range of symbol table entries.

+t *index* Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **-t** option.

-u Underline the name of the file for emphasis.

-v Dump information in symbolic representation rather than numeric (for example, C_STATIC instead of 0X02). This *modifier* can be used with all the above options except **-o** and **-s**.

-z *name,number*
Dump line number entry or range of line numbers starting at *number* for the named function.

+z *number* Dump line numbers starting at either function *name* or *number* specified by **-z**, up to *number* specified by **+z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **-z** option may be replaced by a blank.

dump attempts to format the information in a meaningful way, printing certain information in character, hexadecimal, octal, or decimal representation as appropriate.

See also

a.out(FP), **ar**(FP), **hdr**(XNX)

dumpmsg

generate a message source file

Syntax

dumpmsg [-X] [-C] [*catfile* | -] *msgfile*

Description

dumpmsg reads the formatted message catalogue *catfile* and generates a message source file *msgfile*. To read from standard input or write to standard output use " - " instead of *catfile*.

The following options are recognized:

-X Inhibits production of only X/Open-compatible sources.

-C Converts all non-ASCII characters of the message text to escape sequences representing the character.

Exit status

dumpmsg returns 0 if successful. The following values are returned on error:

1 fatal error occurred.

2 internal 'bug' occurred.

10 encountered warnings.

11 encountered errors (and maybe warnings).

Application usage

A text editor may be used to make leading or trailing blanks visible.

The following shell script using **sed**(C) dumps the messages of message catalogue *chmod.cat*, adds the double-quote character (") to the dumped messages, surrounds all messages with double-quotes, and copies the processed data to standard output:

```
dumpmsg -X chmod.cat - | { echo \$quote \" ; \
sed -e 's/^\([0-9][0-9]*[_\t]\)\(.*\)/\1"\2"/' ; }
```

In the above example, "_" and "\t" represent the space character and the tab character, respectively.

gencat and **dumpmsg** are capable of handling message catalogs with variable-length filenames in their file headers. To convert old message catalogs to this format, use the following example as a model:

```
dumpmsg old.cat new.msg
gencat new.cat new.msg
```

See also

gencat(CP), **mar**(CP)

Standards conformance

This utility is specific to this implementation.

findstr

find strings in C source code

Syntax

findstr [**-b**] *file* ...

findstr [**-b**] **-p** [*cpp_option*] *file* ...

Description

findstr examines files of C source code for string constants, which are written on standard output (one per line in the form of a string list):

```
filename pos len "...message text..."
```

where *filename* is the name of the C source file, *pos* is the character position of the message within the source file and *len* is the message length in bytes.

findstr(CP) assumes a syntactically correct C preprocessed source input file.

In the case of C preprocess request (that is, with the **-p** option), a preprocessed, *.i*-suffixed file is first created in the C source file directory before the the *string list* is generated.

findstr recognizes the following options:

-b Source file directory name truncation within the string file.

-p *cpp_option*
Preprocess source file before performing find. Arguments following **-p** are interpreted as **cpp**(CP) options.

Return values

0 on success

1 on failure

Notes

The output from **findstr**(CP) is normally used by **insertmsg**(CP) to create a new source file with message catalogue information.

See also

gencat(CP), **insertmsg**(CP)

gencat

generate a formatted message catalogue

Syntax

gencat [**-X**] [*catfile* | **-**] [*msgfile* | **-**] [*msgfile* ...]

Description

gencat merges the message text source file(s) *msgfile* into a formatted message catalogue *catfile*. *catfile* will be created if it does not already exist. If *catfile* does already exist its messages will be included in the new *catfile*. If set and message numbers collide, the new message-text defined in *msgfile* will replace the old message text currently contained in *catfile*. **gencat** will complain about formats not defined in the *X/Open Portability Guide*.

The following options are recognized by **gencat**:

- Use standard output or standard input in the place of (respectively) *catfile* or *msgfile*.

-X Inhibits the check for portability to X/Open.

Exit status

gencat returns "0" on success or "1" on failure.

Application usage

Message catalogues produced by **gencat** are binary encoded, meaning that their portability cannot be guaranteed between different types of machines. Therefore, just as C programs need to be recompiled for each type of machine, message catalogues must be recreated via **gencat**.

gencat and **dumpmsg** are capable of handling message catalogs with variable-length filenames in their file headers. To convert old message catalogs to this format, use the following example as a model:

```
dumpmsg old.cat new.msg
gencat new.cat new.msg
```

See also

dumpmsg(CP), **mar**(CP)

Standards conformance

The **-X** option is specific to this implementation.

gencc

create a front-end to the rcc command

Syntax

gencc

Description

The **gencc** command is an interactive command designed to create a front-end to the **rcc** command. Since hard-coded path names have been eliminated from the **rcc**, it is possible to move pieces of the compilation system to new locations without recompiling the system. The new locations of moved pieces can be specified through the **rcc** option **-Y**. However, it is inconvenient to supply **-Y** options with every invocation of **rcc**. Further, if a system administrator moves pieces of the compilation system, such movement should be invisible to users.

The front-end which **gencc** generates is a one-line shell script which calls **rcc** with the proper **-Y** options. The front-end will also pass all user-supplied options to **rcc**.

gencc prompts for the location of each tool and directory which can be respecified by **rcc**'s **-Y** option. If no location is specified, it assumes that that piece of the system has not been relocated. After all the locations have been prompted for, **gencc** creates the front-end to the **rcc** command.

gencc creates the front-end in the current working directory and gives the file the name **rcc**. Thus, **gencc** cannot be run in the same directory containing the actual **rcc** command. If the system administrator has redistributed the compilation system the actual **rcc** command should be placed somewhere which is not typically in a user's PATH (for example, */lib*). This will prevent users from accidentally invoking the **cc** command without using the front-end.

Files

front-end to rcc

Notes

The **gencc** command does not produce any warnings if a tool or directory does not exist at the specified location. Also, **gencc** does not actually move any files to new locations.

See also

rcc(CP)

get

get a version of an SCCS file

Syntax

get [-r*SID*] [-c*cutoff*] [-i*list*] [-x*list*] [-w*string*] [-a*seq-no.*] [-k] [-e] [-l[p]] [-p] [-m] [-n] [-s] [-b] [-g] [-t] *file*...

Description

The **get** command generates an ASCII text file from each named SCCS file according to the specifications given by its arguments. The arguments may be specified in any order, but all arguments apply to all named SCCS files. If a directory is named, **get** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by removing the leading *s.*.

Each of the arguments is explained below as though only one SCCS file is to be processed, but the effects of any argument applies independently to each named file.

-r*SID* The SCCS identification string (*SID*) of the version (delta) of an SCCS file to be retrieved. Table 1 shows what version of an SCCS file is retrieved as a function of the *SID* specified. Also the *PID* of the version to be eventually created by **delta**(CP) if the **-e** option is used.

-c*cutoff* The cutoff date and time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **-c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows you to specify a *cutoff* date in the form: "**-c77/2/2 9:22:25**". Note that this implies that you may use the %E% and %U% identification keywords (see below) for nested **get**s.

```
get "-c%E% %U%" s.file
```

-i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1.

-x*list* A *list* of deltas to be excluded in the creation of the generated file. See the **-i** option for the *list* format.

-e Indicates that the **get** is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of **delta**(CP). The **-e** option used in a **get** for a particular version (*SID*) of the SCCS file prevents further **gets** for editing on the same *SID* until **delta** is executed or the **j** (joint edit) flag is set in the SCCS file (see **admin**(CP)). Concurrent use of **get -e** for different *SID*s is always allowed.

If the *g-file* generated by **get** with an **-e** option is accidentally ruined in the process of editing it, it may be regenerated by re-executing the **get** command with the **-k** option in place of the **-e** option.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see **admin**(CP)) is enforced when the **-e** option is used.

-b Used with the **-e** option to indicate that the new delta should have an *SID* in a new branch as shown in Table 1. This option is ignored if the **b** flag is not present in the file (see **admin**(CP)) or if the retrieved **delta** is not a leaf **delta**. (A leaf **delta** is one that has no successors on the SCCS file tree.)

Note: A branch **delta** may always be created from a non-leaf **delta**. Partial *SID*s are interpreted as shown in the "SID Retrieved" column of Table 1.

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **-k** option is implied by the **-e** option.

-l[p] Causes a delta summary to be written into an *l-file*. If **-lp** is used, then an *l-file* is not created; the delta summary is written on standard output instead.

-p Causes the text retrieved from the SCCS file to be written on standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **-s** option is used, in which case it disappears.

-s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

-m Causes each text line retrieved from the SCCS file to be preceded by the *SID* of the delta that inserted the text line in the SCCS file. The format is: *SID*, followed by a horizontal tab, followed by the text line.

-n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** options are used, the format is: %M% value, followed by a horizontal tab, followed by the **-m** option generated format.

-g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular *SID*.

-t Used to access the most recently created delta in a given release (for example, **-r1**), or release and level (for example, **-r1.2**).

-w *string* Substitute *string* for all occurrences of %W% when getting the file.

-a*seq-no*. The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(FP)). This option is used by the **comb**(CP) command; it is not a generally useful option. If both the **-r** and **-a** options are specified, only the **-a** option is used. Care should be taken when using the **-a** option in conjunction with the **-e** option, as the *SID* of the delta to be created may not be what you expect. The **-r** option can be used with the **-a** and **-e** options to control the naming of the *SID* of the delta to be created.

For each file processed, **get** responds (on standard output) with the *SID* being accessed and with the number of lines retrieved from the SCCS file.

If the **-e** option is used, the *SID* of the delta to be made appears after the *SID* accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **-i** option is used, included deltas are listed following the notation "Included"; if the **-x** option is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	-b Option Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	-	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the *SID*, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (for example, maximum branch number plus one) of level L within release R". Note that if the *SID* specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the **first** delta in a *new* release.

\# Successor.

† The **-b** option is effective only if the **b** flag (see **admin**(CP)) is present in the file. An entry of - means "irrelevant".

‡ This case applies if the **d** (default *SID*) flag is *not* present in the file. If the **d** flag is present in the file, then the *SID* obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

Identification keywords

Identifying information is inserted into the text retrieved from the SCCS file by replacing identification keywords with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	Value
%M%	Module name: either the value of the **m** flag in the file (see **admin**(CP)), or if absent, the name of the SCCS file with the leading *s.* removed.
%I%	SCCS identification (*SID*) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (*YY/MM/DD*).
%H%	Current date (*MM/DD/YY*).
%T%	Current time (*HH:MM:SS*).
%E%	Date newest applied delta was created (*YY/MM/DD*).
%G%	Date newest applied delta was created (*MM/DD/YY*).
%U%	Time newest applied delta was created (*HH:MM:SS*).
%Y%	Module type: value of the **t** flag in the SCCS file (see **admin**(CP)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see **admin**(CP)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by **what**(CP).

%W% A shorthand notation for constructing **what**(CP) strings for system program files. %W% = %Z%%M%<horizontal-tab>%I%

%A% Another shorthand notation for constructing **what**(CP) strings for non-UNIX-type system program files.
%A% = %Z%%Y% %M% %I%%Z%

Several auxiliary files may be created by **get**. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form *s.module-name*; the auxiliary files are named by replacing the leading *s* with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the *s.* prefix. For example, *s.xyz.c*, the auxiliary file names would be *xyz.c*, *l.xyz.c*, *p.xyz.c*, and *z.xyz.c*, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **-p** option is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the **get**. It is owned by the real user. If the **-k** option is used or implied, its mode is 644; otherwise its mode is 444. Only the real user needs write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **-l** option is used; its mode is 444 and it is owned by the real user. Only the real user needs write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied; * otherwise.

b. A blank character if the delta was applied or was not applied and ignored; * if the delta was not applied and was not ignored.

c. A code indicating a "special" reason why the delta was or was not applied:
"I": Included.
"X": Excluded.
"C": Cut off (by a **-c** option).

d. Blank.

e. SCCS identification (*SID*).

f. Tab character.

g. Date and time (in the form *YY/MM/DD HH:MM:SS*) of creation.

h. Blank.

i. Login name of person who created **delta**.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a **get** with a **-e** option along to **delta**. Its contents are also used to prevent a subsequent execution of **get** with a **-e** option for the same *SID* until **delta** is executed or the joint edit flag, **j**, (see **admin**(CP)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten *SID*, followed by a blank, followed by the *SID* that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the **get** was executed, followed by a blank and the **-i** option argument if it was present, followed by a blank and the **-x** option argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta *SID*.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (for example, **get**) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of **get**. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created with mode 444.

Files

g-file	Existed before the execution of **delta**; removed after completion of **delta**.
p-file	Existed before the execution of **delta**; may exist after completion of **delta**.
q-file	Created during the execution of **delta**; removed after completion of **delta**.
x-file	Created during the execution of **delta**; renamed to SCCS file after completion of **delta**.
z-file	Created during the execution of **delta**; removed during the execution of **delta**.
d-file	Created during the execution of **delta**; removed after completion of **delta**.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

Notes

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the **-e** option is used.

See also

admin(CP), **delta**(CP), **prs**(CP), **what**(CP)

Standards conformance

get is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

help

Asks for help about SCCS commands

Syntax

help [*args*]

Description

help finds information to explain a message from an SCCS command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, **help** will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names. There are the following types of arguments:

type 1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, **ge6**, for message 6 from the **get** command).

type 2 Does not contain numerics (as a command, such as **get**)

type 3 Is all numeric (for example, **212**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

Files

/usr/lib/help Directory containing files of message text.

i286emul

emulate 80286

Syntax

i286emul [*arg* ...] *prog286*

Description

i286emul is an emulator that allows programs from System V Release 2 or Release 3 on the Intel 80286 to run on System V Release 3 on the Intel 80386.

The system recognizes an attempt to **exec**(S) a 286 program, and automatically **execs** the 286 emulator with the 286 program name as an additional argument. It is not necessary to specify the **i286emul** emulator on the command line. The 286 programs can be invoked using the same command format as on the 286 System V.

i286emul reads the 286 program's text and data into memory and maps them through the LDT (via **sysi86**(S)) as 286 text and data segments. It also sets callgate 89 in the GDT (which is used by 286 programs for system calls) to point to a routine in **i286emul**. **i286emul** starts the 286 program by jumping to its entry point.

When the 286 program attempts to do a system call, **i286emul** takes control. It does any conversions needed between the 286 system call and the equivalent 386 system call, and performs the 386 system call. The results are converted to the form the 286 program expects, and the 286 program is resumed.

The following are some of the differences between a program running on a 286 and a 286 program using **i286emul** on a 386:

- A 286 program under **i286emul** always has 64k in the stack segment if it is a large-model process, or 64k in the data segment if it is a small-model process.
- System calls and signal handling use more space on the stack under **i286emul** than it does on a 286.
- Attempts to unlink or write on the 286 program will fail on the 286 with ETXTBSY. Under **i286emul**, they will not fail.
- **ptrace**(S) is not supported under **i286emul**.
- The 286 program must be readable for the emulator to read it.

Files

/bin/i286emul
: The emulator must have this name and be in */bin* if it is to be automatically invoked when exec(S) is used on a 286 program.

Notes

The signal mechanism under the emulator is the System V release 2 signal mechanism rather than the System V release 3 mechanism.

iconv

international codeset conversion

Syntax

iconv [-h] [-d] -f *fromcodeset* -t *tocodeset* [*file*...]

Description

The **iconv** command converts the encoding of characters in *file* from *fromcodeset* to *tocodeset* and writes the results on standard output. For example:

```
iconv -f IS8859 -t IS6937 datafile
```

converts the file *datafile* from **IS8859** encoding to **IS6937** encoding, reads the conversion tables from *<path-prefix>/conv/IS8859_IS6937*, and writes the results to standard output.

The conversion tables are held in a file whose name and path are constructed like this:

<path-prefix>/conv/fromcodeset_tocodeset

where *<path-prefix>* defaults to */usr/lib/nls* if and only if the environmental variable **ICONV** is not set.

Each line in the conversion table specifies a conversion from the first string to the second string on the same line. Spaces and tabs are used to delimit tokens, the \ character is used to escape the next character and to introduce (exactly) three digit octal constants. Blank lines and lines starting with a hash (#) are introduced.

```
#       a sample conversion table
#       fromcode     tocode

        \#          wasahash
        \040        wasablank
        string       another\040string
```

Other available options are:

-h Prints a usage message on standard error.

-d Deletes any characters which are not listed in the conversion table (that is, invalid characters). The default is to pass through any invalid characters.

Notes

As the name of the file holding the conversion rules is made with the characters of ***fromcodeset, tocodeset*** and the character " _ ", there may be truncation of some significant characters of the file name in order to conform to the maximum file name length {NAME_MAX}.

Files

$ICONV/conv/***fromcodeset_tocodeset***

insertmsg

separate strings from program logic

Syntax

insertmsg -m [*message_number*] **-v** [*strfile*] . . .

Description

Given a list of C strings contained in input file *strfile* (which is assumed to be a string list file produced by **findstr**(CP)), **insertmsg**(CP) will prepare the identified C source files for message catalogue localization. This is achieved by replacing all string constants (as found in *strfile)* with calls to the **catgets()** subroutine, which retrieves localized files from a natural message catalogue at run time. A new version of each C source file is produced with the characters **nl_** prepended to the original file names (received from *strfile*). In addition to calls of **catgets()**, these new source files will also have the following lines inserted:

```
# include <nl_types.h>
#define  NL_SETN  setnum
extern   nl_catd  _m_catd;
extern   char     *catgets();
```

insertmsg does not perform insertion of message catalogue accessing functions. In addition to creating a new source file, **insertmsg** writes a message source file on the standard output, which can be used as the input to **gencat**(CP), or for natural language localizations of the message text. **insertmsg** does not check string file syntax.

insertmsg recognizes the following options:

-m *message_number*
Set the first message number to be *message_number* (the default value is 1 if **-m** is not present).

-v Set verbose mode.

strfile is a data file containing lines in the format:

filename pos len "...message text..."

Example

Given an input file, *test.c*, the command

```
findstr test.c > strfile.test
```

will generate a string list for *test.c*. The command

```
insertmsg strfile.test
```

will generate a new file *nl_test.c* with **catgets()** subroutines, and output the string list (starting from 1, unless **-m** is used) used in the catalogue on standard output.

Return values

0 on success

1 on failure

Warnings

Possible truncation of the ***nl_file*** base filename.

See also

findstr(CP), **gencat**(CP)

ld

invokes the link editor

Syntax

ld [*options*] *filename* [[*options*] *filename*]

Description

The **ld** command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. It creates an executable program by combining one or more object files and copying the executable result to the file *a.out*. The *filename* must name an object or library file. By convention these names have the *.o* (for object) or *.a* (for archive library) extensions. If more than one name is given, the names must be separated by one or more spaces. If any input file, *filename*, is not an object file, **ld** assumes it is either an archive library or a text file containing link editor directives. By default, the file *a.out* is executable if no errors occurred during the load. If errors occur while linking, **ld** displays an error message; the resulting *a.out* file is unexecutable.

ld concatenates the contents of the given object files in the order given in the command line. Library files in the command line are examined only if there are unresolved external references encountered from previous object files.

The library is searched iteratively to satisfy as many references as possible and only those routines that define unresolved external references are concatenated. The library (archive) symbol table (see *ar*(FP)) is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is functionally unimportant, unless there are multiple library members defining the same external symbol. The library may be either a relocatable archive library or a shared library. Object and library files are processed at the point they are encountered in the argument list, so the order of files in the command line is important. In general, all object files should be given before library files. **ld** sets the entry point of the resulting program to the beginning of the first routine.

ld should be invoked using the **cc**(CP) command instead of directly. **cc** invokes **ld** as the last step of compilation, providing all the necessary C language support routines. Invoking **ld** directly is not recommended since failure to give command line arguments in the correct order can result in errors.

Generating COFF vs. x.out binaries

When **ld** is called, it scans all the object files that are to be linked. It determines whether an object file is in *x.out* or COFF format and executes the appropriate binaries in each case. Since this utility relies on the files in */lib/coff* and */lib/xout*), changing or moving any of those files results in an error. If all the object files are COFF objects, then the resulting binary will be in COFF format. If any of the object files to be linked are in *x.out* format, any COFF modules in the group will be converted to *x.out* and the resulting binary will be in *x.out* format.

Common options

The following options are recognized by **ld**, and are common to producing both COFF and *x.out* binaries. Refer to the sections "Linking COFF Binaries" and "Linking x.out Binaries" for format-specific options.

-m ***mapfile***
Produce a map of the input/output sections and store it in ***mapfile***. If no ***mapfile*** is specified, the map is sent to standard output.

-o ***name***
Sets the executable program filename to ***name*** instead of *a.out*.

-r XENIX Version: Invokes the incremental linker, */lib/ldr*, with the arguments passed to **ld** to produce a relocatable output file.

-r AT&T Version: retains relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent **ld** run. The link editor will not complain about unresolved references, and the output file will not be executable.

-s Strips line number entries and symbol table information from the output object file.

-u ***symbol***
Designates the specified ***symbol*** as undefined. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the **ld** line is significant; it must be placed before the library which defines the symbol.

-V Outputs a message giving information about the version of **ld** being used.

Linking COFF binaries

The following options are used for linking COFF binaries:

-a Create an absolute (executable) file. This is the default except when the **-r** option is also used. If **-r** is used, then **-a** allocates memory for common symbols.

-e *epsym*
Set the default entry point address for the output file to be that of the symbol *epsym*.

-f *fill*
Set the default fill pattern for "holes" within an output section as well as initialized *.bss* sections. The argument *fill* is a two-byte constant.

-l*x* Search a library *libx.a*, where *x* is up to nine characters. A library is searched when its name is encountered, so the placement of a **-l** is significant. By default, libraries are located in *LIBDIR* or *LLIBDIR*.

-L *dir*
Change the algorithm of searching for *libx.a* to look in *dir* before looking in *LIBDIR* and *LLIBDIR*. This option is effective only if it precedes the **-l** option on the command line.

-M Output a message for each multiply-defined external definition.

-N Put the text section at the beginning of the text segment rather than after all header information, and put the data section immediately following text in the core image.

-strict
Suppress the generation of the symbols **edata**, **etext** and **end** which are aliases, used for compatibility with previous releases, to the symbols **_edata**, **_etext** and **_end**. This prevents name space pollution when compiling in any of ANSI, POSIX, or XOPEN modes (see **-ansi**, **-posix** and **-xpg3** options to the cc(CP)command.

-t Turn off the warning about multiply-defined symbols that are not the same size.

-VS *num*
Use *num* as a decimal version stamp identifying the *a.out* file that is produced. The version stamp is stored in the optional header.

-x Do not preserve local symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.

-Y[LU],*dir*
Change the default directory used for finding libraries. If **L** is specified, the first default directory which **ld** searches, *LIBDIR*, is replaced by ***dir***. If **U** is specified and **ld** has been built with a second default directory, *LLIBDIR*, then that directory is replaced by ***dir***. If **ld** was built with only one default directory and **U** is specified a warning is printed and the option is ignored.

-z Do not bind anything to address zero. This option will allow runtime detection of null pointers.

Linking x.out binaries

The user must make sure that the most recent library versions have been processed with **ranlib**(XNX) before linking. Library files for *x.out* format binaries must be in **ranlib**(XNX) format, that is, the first member must be named __.SYMDEF, which is a dictionary for the library. **ld** compares the modification dates of the library and the __.SYMDEF entry, so if object files have been added to the library since __.SYMDEF was created, the link may result in an "invalid object module" that cannot run.

The following options are recognized by **ld** for linking *x.out* binaries:

-A *hex-num*
Creates a standalone program whose expected load address (in hexadecimal) is ***hex-num***. This option sets the absolute flag in the header of the *a.out* file. Such program files can only be executed as standalone programs. Options **-A** and **-F** are mutually exclusive.

-B *hex-num*
Sets the text selector bias to the specified hexadecimal number.

-c *num*
Alters the default target CPU in the *x.out* header. ***num*** can be 0, 1, 2, or 3 indicating 8086, 80186, 80286, or 80386 processors, respectively. The default value is 3. Note that this option only alters the default; if object modules containing code for a higher numbered processor are linked, then that will take precedence over the default.

-C Causes the link editor to ignore the case of symbols.

-D *hex-num*
Sets the data selector bias to the specified hexadecimal number.

-F *hex-num*
Sets the size of the program stack to ***hex-num*** bytes where ***hex-num*** is a hexadecimal number. This option is ignored for 80386 programs, which have a variable-sized stack. By default 8086 programs have a variable stack located at the top of the first data segment, and 80286 programs have a fixed size 4096 byte stack. The **-F** option is incompatible with the **-A** option.

-g Includes symbolic information for **Codeview**.

-i Creates separate instruction and data spaces for small model programs. When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and shared by all users executing the file.

-La Sets advisory file locking. Advisory locking is used on files with access modes that do not require mandatory locking.

-Lm Sets mandatory file locking. Mandatory file locking is used on files that cannot be opened by more than one process at a time.

-M*x* Specifies the memory model. *x* can have the following values:

s	small
m	medium
l	large
h	huge
e	mixed

-n ***num***
Truncates symbols to the length specified by ***num***.

-N ***hex-num***
Sets the pagesize to ***hex-num*** (which should be a multiple of 200 hexadecimal or 512 decimal). The default is 400 hexadecimal (1024 decimal) for 80386 programs. 8086/80186/80286 programs do not normally have page-aligned *x.out* files and the default for these is 0.

-P Disables packing of segments

-R Ensures that the relocation table is of non-zero size. Important for 8086 compatibility.

-Rd ***hex-num***
Specify the data segment relocation offset (80386 only).

-Rt ***hex-num***
Specify the text segment relocation offset (80386 only).

-S ***num***
Sets the maximum number of segments to ***num***. If no argument is given, the default is 128.

Files

/bin/ld	
LIBDIR/lib*x*.a	libraries
LLIBDIR/lib*x*.a	libraries
a.out	output file
LIBDIR	usually */lib*
LLIBDIR	usually */usr/lib*

Notes

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the program's address space.

- When the link editor is called through **cc**(CP), a startup routine is linked with the user's program. This routine calls **exit( )** (see **exit**(S)) after execution of the main program. If the user calls the link editor directly, then the user must insure that the program always calls **exit( )** rather than falling through the end of the entry routine.

The symbols **etext**, **edata**, and **end** (see **end**(S)) are reserved and are defined by the link editor. It is incorrect for a user program to redefine them. The following only applies to the linking of COFF binaries. The "_" forms are always defined, the non "_" forms are also defined in non-strict modes.

If the link editor does not recognize an input file as an object file or an archive file, it will assume that it contains link editor directives and will attempt to parse it. This will occasionally produce an error message complaining about "syntax errors".

Arithmetic expressions may only have one forward referenced symbol per expression.

If you are using XENIX binaries, please refer to the **ld**(XNX) manual page for information on the appropriate usage with XENIX binaries.

See also

a.out(FP), **ar**(FP), **as**(CP), **cc**(CP), **end**(S), **exit**(S), **masm**(CP), **mkshlib**(CP), **ranlib**(XNX)

Standards conformance

ld is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

lex

a lexical-analyzer generator

Syntax

lex [**-tvn**] [*specifications-files*] ...

Description

The **lex** command generates C code which implements a lexical analyzer—a routine which reads input text and separates it into tokens. The **lex** input specifications (that is, all the input files concatenated together) consist of three sections: declarations; a rules section consisting of regular expressions (patterns) which define the token classes and usually some C code to be executed when tokens are found; and subroutines. The first and third sections are optional. The sections are delimited by the sequence %%. The rules section *must* start with this delimiter.

lex generates a file of C code called *lex.yy.c*. This file must be compiled by the C compiler and linked with a **main** routine. The program should be linked with the **lex** library, using the **-ll** option to **cc** or **ld**. This library supplies a **main** routine. The lexical analyzer routine produced is called **yylex**. This routine reads its input and, when a token is recognized, executes the code associated with the token class. The default action is to write the token to the standard input. The string matched by the regular expression defining the token class is placed in **yytext**, a character array. The variable **yyleng** gives the length of this array. This value of **yytext** may be copied into an external array to make it available to other routines.

The regular expressions understood by **lex** contain many of the usual operators and special characters. The following table summarizes these:

string	the literal *string*
*	zero or more occurrences of the preceding pattern
+	one or more occurrences of the preceding pattern
?	zero or one occurrences of the preceding pattern
.	any single character
|	alternation
( )	used for grouping
~	beginning of an input line
^	end of an input line
pattern{*n*,*m*}	*n* to *m* occurrences of *pattern*
pattern{*n*}	*n* occurrences of *pattern*
[*string*]	any character in *string*
[^*string*]	any character not in *string*
[*char1-char2*]	any character in the range *char1-char2*

Special characters can be escaped or quoted if they are to be used as ordinary characters. The standard C escape sequences are understood. Regular expressions may be concatenated. The character " / " in an expression indicates that the expression that follows must be matched in order for the token to be matched; only the part of the expression up to the slash is placed in **yytext**.

The declarations section of a **lex** input file may contain variable declarations, **#include** statements, and abbreviations for regular expressions. The subroutines section contains user-defined functions used by the lexical analyzer.

Any line beginning with a blank is assumed to contain only C text and is copied to the file *lex.yy.c*; if it is in the declarations section, it is copied into the external definition area of the *lex.yy.c* file. Variable declarations and **#include** statements should be placed in a section delimited by %{ and %}. Abbreviations consist of a symbol on the left of the line and its replacement text to the right. When abbreviations are used they are surrounded by curly braces, {}.

Three I/O routines are defined: **input**( ) reads a character; **unput**(*c*) returns a character to the input stream; **output**(*c*) outputs a character. These routines may be redefined by the user.

Other built-in routines include the following: **REJECT**, on the right side of the rule, causes the match to be rejected and the next suitable match executed; the function **yymore**( ) accumulates additional characters into **yytext**; the function **yyless**(*p*) pushes back the portion of the string matched beginning at position *p*.

The variable names generated by lex all begin with the prefix *yy* or *YY*. Users should avoid defining variables starting with these prefixes.

The lexical analyzer's implementation involves finite state machine; this state machine can be configured in the declarations section. This is done with a declaration of the following form, where *x* is a key letter, and *n* is an integer:

```
%x n
```

The following parameters may be set in this way:

Key letter	Meaning	Default
p	number of positions	2500
n	number of states	500
e	number of parse tree nodes	1000
a	number of transitions	2000
k	number of packed character classes	1000
o	size of output array	3000

The use of one or more of the above automatically causes a summary of statistics to be printed. See **-v** and **-n** options, below.

Options

The options must appear before any files.

-t This causes the generated code to be written to the standard out rather than to *lex.yy.c*

-v Provides a one-line summary of statistics. This is flagged automatically if any finite state machine parameters are set.

-n Suppresses the summary of statistics even if **-v** is turned on.

Multiple files on the command line are concatenated and treated as a single file. If no files are given, standard input is used.

Example

The following is an example of a lex specification. It shows the use of each of the three sections in the input.

```
%{
#include "global.h"
int count;
%}
D        [0-9]
%%
if       {
                printf("IF statement\n");
                count++;
         }
[a-z]+    printf("tag, value %s\n",yytext);
0(D)+      printf("octal number %s\n",yytext);
{D}+       printf("decimal number %s\n",yytext);
"++"       printf("unary op\n");
"+"        printf("binary op\n");
"/*"       skipcommnts();
%%
 skipcommnts()
 {
        for (;;)
        {
                while (input() != '*')
                        ;
                if (input() != '/')
                        unput(yytext[yyleng-1]);
                else
                        return;
        }
 }
```

See also

yacc(CP)

Standards conformance

lex is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

lint

a C program checker

Syntax

lint [*option*] ... *file* ...

Description

The **lint** command attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

lint is based on the Microsoft compiler **cc**. It is capable of understanding both ANSI and Microsoft syntax. To check programs written for the AT&T compiler **rcc**, use the **rlint** command (see the **rlint**(CP) manual page).

Arguments whose names end with *.c* are taken to be C source files. Arguments whose names end with *.ln* are taken to be the result of an earlier invocation of **lint** with either the **-c** or the **-o** option used. The *.ln* files are analogous to *.o* (object) files that are produced by the **cc**(CP) command when given a *.c* file as input. Files with other suffixes are warned about and ignored.

The **lint** command will take all the *.c*, *.ln*, and *llib-lx.ln* (specified by **-l***x*) files and process them in their command line order. By default, **lint** appends the standard C **lint** library (*llib-lc.ln*) to the end of the list of files. However, if the **-p** option is used, the portable C **lint** library (*llib-port.ln*) is appended instead. When the **-c** option is not used, the second pass of **lint** checks this list of files for mutual compatibility. When the **-c** option is used, the *.ln* and the *llib-lx.ln* files are ignored.

Any number of **lint** options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

-a Suppress complaints about assignments of long values to variables that are not long.

-b Suppress complaints about **break** statements that cannot be reached. (Programs produced by **lex** or **yacc** will often result in many such complaints.)

-u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running **lint** on a subset of files of a larger program.)

-v Suppress complaints about unused arguments in functions.

-x Do not report variables referred to by external declarations but never used.

The following arguments alter **lint**'s behavior:

-c Cause **lint** to produce a *.ln* file for every *.c* file on the command line. These *.ln* files are the product of **lint**'s first pass only, and are not checked for inter-function compatibility.

-l*x* Include additional **lint** library *llib-lx.ln*. For example, you can include a **lint** version of the math library *llib-lm.ln* by inserting **-l***m* on the command line. This argument does not suppress the default use of *llib-lc.ln*. These **lint** libraries must be in the assumed directory. This option can be used to reference local **lint** libraries and is useful in the development of multi-file projects.

-n Do not check compatibility against either the standard or the portable **lint** library.

-o *lib*
Cause **lint** to create a **lint** library with the name *llib-llib.ln*. The **-c** option nullifies any use of the **-o** option. The lint library produced is the input that is given to **lint**'s second pass. The **-o** option simply causes this file to be saved in the named **lint** library. To produce a *llib-llib.ln* without extraneous messages, use of the **-x** option is suggested. The **-v** option is useful if the source file(s) for the **lint** library are just external interfaces (for example, the way *llib-lc* is written). These option settings are also available through the use of "lint comments" (see below).

-p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

The **-D**, **-U**, and **-I** options of **cpp**(CP) and the **-g** and **-O** options of **cc**(CP) are also recognized as separate arguments. The **-g** and **-O** options are ignored, but, by recognizing these options, **lint**'s behavior is closer to that of the **cc**(CP) command. Other options are warned about and ignored.

The symbols "__lint" and "lint" should be thought of as reserved words for all code that you plan to check with **lint**. The default preprocessor symbol "__lint" is defined to allow certain questionable code to be altered or removed for **lint**. The preprocessor symbol "lint" is also currently defined to allow backward compatibility.

Certain conventional comments in the C source will change the behavior of **lint**:

/*ARGSUSED*/
Turns on the **-v** option for the next function.

/*LINTLIBRARY*/
Placed at the beginning of a file turns off complaints about unused functions and function arguments in this file. This is equivalent to using the **-v** and **-x** options.

/*NOTREACHED*/
Placed at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like **exit**(S).)

/*VARARGS*n*/
Suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

The **lint** command produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the **-c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the **-c** and the **-o** options allows for incremental use of **lint** on a set of C source files. Generally, one invokes **lint** once for each source file with the **-c** option. Each of these invocations produces a *.ln* file for each *.c* file, and prints all messages that are about just that source file. After all the source files have been separately run through **lint**, it is invoked once more (without the **-c** option), listing all the *.ln* files with the needed **-l***x* options. This will print all the interfile inconsistencies. This scheme works well with **make**(CP); it allows **make** to be used to **lint** only the source files that have been modified since the last time the set of source files were **linted**.

Files

LLIBDIR	The directory where the **lint** libraries specified by the **-l***x* option must exist, usually */usr/lib*.
LLIBDIR/lint[12]	First and second passes.
LLIBDIR/llib-lc.ln	Declarations for C Library functions (binary format; source is in *LLIBDIR/llib-lc*).

LLIBDIR/llib-port.ln	Declarations for portable functions (binary format; source is in *LLIBDIR/llib-port*).
LLIBDIR/llib-lm.ln	Declarations for Math Library functions (binary format; source is in *LLIBDIR/llib-lm*).
TMPDIR/*lint*	Temporary files.
TMPDIR	Usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**() in **tmpnam**(S)).

Notes

exit(S), **setjmp**(S), and other functions that do not return are not understood; this causes various incorrect warnings.

See also

cc(CP), **cpp**(CP), **make**(CP)

Standards conformance

lint is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

list

produce C source listing from a common object file

Syntax

list [-V] [-h] [-F *function*] *source-file* ... [*object-file*]

Description

The **list** command produces a C source listing with line number information attached. If multiple C source files were used to create the object file, **list** will accept multiple file names. The object file is taken to be the last non-C source file argument. If no object file is specified, the default object file, *a.out*, will be used. This utility operates only on COFF executables. Object files given to **list** must have been compiled with the **-g** option of **cc**(CP).

Line numbers will be printed for each line marked as breakpoint inserted by the compiler (generally, each executable C statement that begins a new line of source). Line numbering begins anew for each function. Line number 1 is always the line containing the left curly brace (" { ") that begins the function body. Line numbers will also be supplied for inner block redeclarations of local variables so that they can be distinguished by the symbolic debugger.

The following options are interpreted by **list** and may be given in any order:

-F*function* List only the named function. The **-F** option may be specified multiple times on the command line.

-h Suppress heading output.

-V Print, on standard error, the version number of the **list** command executing.

Diagnostics

The **list** command will produce the error message "list: name: cannot open" if *name* cannot be read. If the source file names do not end in *.c*, the message is "list: name: invalid C source name". An invalid object file will cause the message "list: name: bad magic" to be produced. If some or all of the symbolic debugging information is missing, one of the following messages will be printed: "list: name: symbols have been stripped, cannot proceed", "list: name: cannot read line numbers", or "list: name: not in symbol table". The following messages are produced when **list** has become confused by **#ifdef's** in the source file: "list: name: cannot find function in symbol table", "list: name: out of sync: too many }", and "list: name: unexpected end-of-file". The error message "list: name: missing or inappropriate line numbers" means that either symbol debugging information is missing, or **list** has been confused by C preprocessor statements.

Notes

Since **list** does not use the C preprocessor, it may be unable to recognize function definitions whose syntax has been distorted by the use of C preprocessor macro substitutions.

See also

as(CP), **cc**(CP), **ld**(CP)

lorder

find ordering relation for an object library

Syntax

lorder *file* ...

Description

The input is one or more object or library archive **files** (see **ar**(CP)). The standard output is a list of pairs of object file or archive member names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by **tsort**(CP) to find an ordering of a library suitable for one-pass access by **ld**(CP). Note that the link editor **ld**(CP) is capable of multiple passes over an archive in the portable archive format (see *ar*(FP)) and does not require that **lorder**(CP) be used when building an archive. The usage of the **lorder**(CP) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing *.o* files.

```
ar -cr library `lorder *.o | tsort`
```

Files

TMPDIR/*symref	temporary files
TMPDIR/*symdef	temporary files

TMPDIR is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**( ) in **tmpnam**(S)).

Note

lorder will accept as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is only a single input file, its suffix must be *.o*.

See also

ar(CP), **ar**(FP), **ld**(CP), **tsort**(CP)

Standards conformance

lorder is conformant with:

AT&T SVID Issue 2.

lprof

display line-by-line execution count profile data

Syntax

lprof [-p] [-s] [-x] [[-I *incdir*]] [[-r *srcfile*]] [-c *cntfile*] [-o *prog*]

lprof -m *file1.cnt file2.cnt* [...*filen.cnt*] [-T] -d *destfile.cnt*

Description

lprof is a tool for dynamic analysis; that is, the analysis of a program at run time. Specifically, **lprof** identifies the most frequently executed parts of source code and parts of code that are never executed.

lprof interprets a profile file (*prog.cnt* by default) produced by the profiled program *prog* (*a.out* by default) that has been compiled with the **-ql** option of **cc**(CP) or **rcc**(CP). This **cc** command option arranges for code to be inserted to record run-time behavior and for data to be written to a file at the end of execution.

By default, **lprof** prints a listing of source files, the names of which are stored in the symbol table of the executable file. Each line preceded by its line number and the number of times it was executed.

Options

The following options may appear singly or be combined in any order:

-p Print listing, each line preceded by the line number and the number of times it was executed (default). This option can be used together with the **-s** option to print both the source listing and summary information.

-s Print summary information of percentage of lines of code executed per function.

-x Instead of printing the execution count numbers for each line, print each line preceded by its line number and a [U] if the line was not executed. If the line was executed, print only the line number.

-I *incdir*
Look for source or header files in the directory *incdir* in addition to the current directory and the standard place for **#include** files (usually */usr/include*). You can specify more than one directory with **-I** on the command line.

-r *srcfile*
Instead of printing all source files, print only those files named in **-r** options (to be used with the **-p** option only). You can specify multiple files with **-r** on one command line.

-c *cntfile*
Use the file *cntfile* instead of *prog.cnt* as the input profile file.

-o *prog*
Profile the program *prog* instead of the name used when creating the profile file. Because the program name stored in the profile file contains the relative path, this option is necessary if the executable file or profile file has been moved.

Merging data files

lprof can also be used to merge data files. The **-m** option must be accompanied by the **-d** option:

-m *file1.cnt file2.cnt* [*filen.cnt*] **-d***destfile.cnt*
Merge the data files *file1.cnt* through *filen.cnt* by summing the execution counts per line, so that data from several runs can be accumulated. The result is written to *destfile.cnt*. The data files must contain profiling data for the same *prog* (see the **-T** option below).

-T Time stamp override. Normally, the time stamps of the executable files being profiled are checked, and data files will not be merged if the time stamps do not match. If **-T** is specified, this check is skipped.

Controlling the run time profiling environment

lprof uses the environment variable **PATH** to locate the correct full pathname of the executable, for example, *../../file* has an explicit pathname, but *file* does not. The environment variable **PROFOPTS** provides run-time control over profiling. When a profiled program is about to terminate, it examines the value of **PROFOPTS** to determine how the profiling data is to be handled.

The environment variable **PROFOPTS** is a comma-separated list of options interpreted by the program being profiled. If **PROFOPTS** is not defined in the environment, then the default action is taken: the profiling data is saved in a file (with the default name, *prog.cnt*) in the current directory. If **PROFOPTS** is set to the null string, no profiling data is saved. The following are the available options:

msg=[yn]
If **msg=y** is specified, a message stating that profile data is being saved is printed to standard error. If **msg=n** is specified, print only profiling error messages. The default is **msg=y**.

merge=[yn]
If **merge=n** is specified, do not merge data files after successive runs. The data file is overwritten after each execution. If **merge=y** is specified, the data will be merged. The merge will fail if the program has been recompiled; the data file will be left in *TMPDIR*. The default is **merge=n**.

pid=[yn]
If **pid=y** is specified, the name of the data file will include the process ID of the profiled program. This allows the creation of different data files for programs calling **fork**(S). If **pid=n** is specified, the default name is used. The default is **pid=n**.

dir=*dirname*
Place the data file in the directory ***dirname*** if this option is specified. Otherwise, the data file is created in the directory that is current at the end of execution.

file=*filename*
Use ***filename*** as the name of the data file in ***dir*** created by the profiled program if this option is specified. Otherwise, the default name is used.

Files

prog.cnt	For profile data.
TMPDIR/*	Temporary files.

TMPDIR is usually */usr/tmp*, but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**( ) in **tmpnam**(S)).

Warnings

For the **-m** option, if ***destfile.cnt*** exists, its previous contents are destroyed.

Optimizing functions may result in the loss of some line number information and may result in code motions, both of which may make **lprof** information unreliable.

See also

cc(CP), **fork**(S), **prof**(CP), **tmpnam**(S)

m4

macro processor

Syntax

m4 [*options*] [*file*] . . .

Description

The **m4** command is a macro processor for use as a front-end for C and other languages. Each input file is processed in order; if no file names are specified, or if the *file* is "-", standard input is read. The processed text is written to standard output.

The options and their effects are as follows:

-e Operate interactively. Interrupts are ignored and the output is unbuffered.

-s Enable line sync output for the C preprocessor (**#line** . . .)

-B*int* Change the size of the push-back and argument collection buffers from the default of 4,096.

-H*int* Change the size of the symbol table hash array from the default 199. The size should be prime.

-S*int* Change the call stack size from the default 100 slots. Macros take three slots, and non-macro arguments take one.

-T*int* Change the token buffer size from the default 512 bytes.

To be effective, these flags must appear before any file names and before any **-D** or **-U** flags:

-D*name*[=*val*] Defines *name* to *val* or to null in *val*'s absence.

-u undefine all built-in, namespace-polluting macros

-U*name* Undefines *name*.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The open parenthesis, "(", must immediately follow the macro name. If the name of a defined macro is not followed by a close parenthesis, ")", it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore "_", where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

m4 makes the following built-in macros available. They may be re-defined, but once this is done, the original meaning is lost. Their values are null unless otherwise stated.

define This macro takes two arguments: the first is the macro name, and the second is its definition. When the macro name appears in the input text, it will be replaced by its definition. The macro definition may be a simple text string or may involve arguments. Arguments are of the form *$n*, where *n* is a digit between 0 and 9. If an occurrence of a macro name is followed by a comma-separated list of values, enclosed in parentheses, then when the macro substitution takes place, the *n*th value in the list replaces the *$n*. **$0** is the name of the macro; missing arguments are replaced by a null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is like **$***, but each argument is quoted (with the current quotes). The replacement text is then re-processed by **m4**

undefine Removes the definition of the macro named in its argument.

defn Returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

pushdef Operates like **define**, but saves any previous definition.

popdef Removes current definition of its argument(s), exposing the previous one, if any.

ifdef If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word **unix** is predefined on some versions of **m4**.

shift Returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

changequote Change quote symbols to the first and second arguments. The symbols may be up to five characters long. **changequote** without arguments restores the original values (that is, ` ´).

changecom Change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

divert **m4** maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. **divert** changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert Causes immediate output of text from diversions named as arguments, or all diversions if no argument is supplied. Text may be "undiverted" into another diversion.

divnum Returns the value of the current output stream.

dnl Reads and discards characters up to and including the next new-line.

ifelse Has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6, and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

incr Returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

decr Returns the value of its argument decremented by 1.

eval Evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

len Returns the number of characters in its argument.

index Returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr Returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit Transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include Returns the contents of the file named in the argument.

sinclude Is identical to **include**, except that it says nothing if the file is inaccessible.

syscmd Executes the system command given in the first argument. No value is returned.

sysval The return code from the last call to **syscmd**.

maketemp Fills in a string of XXXXX in its argument with the current process ID.

m4exit Causes immediate exit from **m4**. Argument 1, if given, is the exit code; the default is 0.

m4wrap Argument 1 will be pushed back at final EOF; for example:

```
m4wrap(`cleanup()')
```

errprint Prints its argument on the diagnostic output file.

dumpdef Prints current names and definitions for the named items or for all if no arguments are given.

traceon With no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

traceoff Turns off trace globally and for any macros specified. Macros specifically traced by **traceon** can be untraced only by specific calls to **traceoff**.

See also

cc(CP), **cpp**(CP).

Standards conformance

m4 is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

make

maintain, update, and regenerate groups of programs

Syntax

make [-f *makefile*] [-piksrnbBetqd] [*names*] [*macro definitions*]

Description

make allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

-f *makefile* Description file name. ***makefile*** is assumed to be the name of a description file.

-p Print out the complete set of macro definitions and target descriptions.

-i Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.

-k Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.

-s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.

-r Do not use the built-in rules.

-n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

-b Compatibility mode for old makefiles; this is the default mode.

-B Turn off compatibility mode for old makefiles.

-e Environment variables override assignments within makefiles.

-t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

-q Question. The **make** command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

Special targets

.DEFAULT If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.PRECIOUS Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT Same effect as the **-s** option.

.IGNORE Same effect as the **-i** option.

make evaluates the dependencies specified in ***makefile*** and executes the associated commands to update one or more targets, specified by ***name***. Targets typically are programs. If no **-f** option is present, then the files *makefile*, *Makefile*, and the Source Code Control System (SCCS) files *s.makefile*, and *s.Makefile* are tried in order. If ***makefile*** is **-**, the standard input is read. More than one **-f** ***makefile*** argument pair may appear.

make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date, that is, they are newer than the target.

makefile contains a sequence of entries that specify dependencies. A dependency has the following syntax:

> ***target1*** [***target2*** ...] **:**[**:**] [***dependency1*** ...] [**;** ***commands*** [# ...]
> [[⟨Tab⟩ ***commands***][# ...]
> ...
>]

(⟨Tab⟩ is the tab character on your keyboard. It is printed here as ⟨Tab⟩, instead of a sequence a blank characters for the purposes of this description.)

Tokens on the first line of an entry are blank-separated. The first non-empty line that does not begin with a tab or # begins a new dependency or macro definition. If the targets are separated from the dependent files by double colon (::), then the targets can appear in several dependencies. Each such dependency will be evaluated separately, using the original last modification time of the target to determine whether the commands are executed.

Shell commands and comments may be continued across lines with the <backslash><new-line> sequence. Everything printed by **make** (except the initial tab) is passed directly to the shell as is. If the last character on a line is a backslash, then the backslash and the new-line character are replaced by a single blank.

Sharp (#) and new-line surround comments.

The following ***makefile*** says that ***pgm*** depends on two files *a.o* and *b.o*, and that they in turn depend on their corresponding source files (*a.c* and *b.c*) and a common file *incl.h*:

```
pgm: a.o b.o
   cc a.o b.o -o pgm

a.o: incl.h a.c
   cc -c a.c

b.o: incl.h b.c
   cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The **SHELL** environment variable can be used to specify which shell **make** should use to execute commands. The default is the users default shell. The first one or two characters in a command can be the following: **-**, **@**, **-@**, or **@-**. If **@** is present, printing of the command is suppressed. If **-** is present, **make** ignores return codes. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is in ***makefile***, or unless the initial character sequence contains a **@**. The **-n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the**MAKEFLAGS** macro under "Environment"). The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate **make**. If the **-i** option is present, or the entry **.IGNORE:** appears in ***makefile***, or the initial character sequence of the command contains **-**, the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name **.PRECIOUS.**

Environment

The environment is read by **make**. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **-e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The **MAKEFLAGS** environment variable is processed by **make** as containing any legal input option (except **-f** and **-p**) defined for the command line. Further, upon invocation, **make** "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **-n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make -n** recursively on a whole software system to see what would

have been executed. This is because the **-n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without executing the commands in the makefile.

If a macro named **VPATH** is assigned a list of colon separated directory names, **make** will search these directories when looking for files with relative path names, in the same way the shells use the **PATH** environment variable.

Include files

If the string **include** appears as the first seven letters of a line in a ***makefile***, and is followed by a blank or a tab, the rest of the line is assumed to be a filename and will be read by the current invocation, after substituting for any macros.

Inference rules

make has a set of internal rules for building files which control the inference of prerequisites. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target. For example, the rule to create a file with suffix *.o* from a file with suffix *.c* is specified as an entry with *.c.o:* as the target and no dependents. Shell commands associated with the target define the rule for making a *.o* file from a *.c* file. You may add rules to this list by simply putting them in the ***makefile***.

Using the inference rules, the first example can be done more briefly.

```
pgm: a.o b.o
cc a.o b.o -o pgm
a.o b.o: incl.h
```

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to **cc**(CP), **lex**(CP), and **yacc**(CP), respectively. Use the **-p** option to list the internal rules.

Macros

Entries of the form ***string1*** = ***string2*** and ***string3*** == ***string4*** are macro definitions. Subsequent appearences of ***$(string1[:subst1=subst2])*** on dependency lines or in shell commands are replaced by ***string2***. Subsequent appearences of ***$(string3[:subst1=subst2])*** anywhere in the description file are replaced by ***string4***. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, newline characters, and beginnings of lines.

The optional ***:subst1=subst2*** is a substitute sequence. The parentheses are optional if a single character macro name is used and there is no substitute sequence. If a substitute sequence is specified, all nonoverlapping occurances of ***subst1*** in the named macro are replaced by ***subst2***.

If *string2* in the single equal sign assignment contains macros, they are not evaluated until *string1* is evaluated in a dependency line or a command line. This saves time and space. However, if two or more substitution operator assignments are done using single equal sign assignments, an error will occur when the macro is evaluated on a dependency line or command line because make does not support nested evaluations of the substitution operator. For example, if the description file is

```
SRC=src/file1.c src/file2.c src/file3.c
TMP=$(SRC:.c=.o)
OBJ=$(TMP:src=obj)

all: $(SRC)

clean:
	rm $(OBJ)
```

and you execute it, requesting **clean** to be updated, an error will occur because **make** will only evaluate the outer substitution operator of **$(OBJ)** resulting in the shell executing **rm $(SRC:src=obj)**. If the double equal sign assignment were used in the **TMP** assignment, then **TMP** would immediately be evaluated, removing the inner substition operator from **OBJ** and the **clean** target would evaluate correctly.

Internal macros

There are five internally maintained macros that are useful for writing rules for building targets.

$*** **$* stands for the filename part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** **$@ stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** **$< is only evaluated for inference rules or the **.DEFAULT** rule. It is the module that is out-of-date with respect to the target (that is, the "manufactured" dependent file name). Thus, in the *.c.o* rule, the **$<** macro would evaluate to the *.c* file. An example for making optimized *.o* files from *.c* files is:

```
.c.o:
	cc -c -O $*.c
```

or:

```
.c.o:
	cc -c -O $<
```

$?** **$? is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% $% is only evaluated when the target is an archive library member of the form *lib(file.o)*. In this case, $@ evaluates to *lib* and $% evaluates to the library member, *file.o*.

$! $! evaluates to target name, if the target is the default target or if it was requested on the command line, or to the list of targets which were called as the result of dependency relationships. For example, suppose the description file is:

```
p q: r
        @echo $!
r: s
        @echo $!
s:
        @echo $!
```

If this file were called requesting 'q' be updated, then the following would be printed:

```
s r q
r q
q
```

The line `s r q` is printed by **echo** following the s target, the line `r q` is printed by **echo** following the r target, and the line `q` is printed by **echo** following the p q targets.

All of the internal macros can have an alternative form. When an uppercase **D** or **F** is appended to any of the macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, *./* is generated.

Suffixes

The **make** program uses an internal table of rules to learn how to transform a file with one suffix into a file with another. Certain names (for example, those ending with *.o*) have inferrible prerequisites such as *.c*, *.s*, etc. These transformation rules names are the concatenation of two suffixes. For example the name of the rule to transform a *.r* file to a *.o* is *.r.o*. If a target is whose suffix corresponds to one of the internal rules is listed in a makefile and no explicit commands are given in its dependency, or if a file is a dependent of another target and no target is explicitly given for it but its suffix matches one of the internal rule suffixes, then the commands for the corresponding internal rule are used to make the target.

To print out the rules compiled into **make** on any machine in a form suitable for reuse, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

A tilde in the above rules refers to an SCCS file (see *sccsfile*(FP)). Thus, the rule *.c~.o* would transform an SCCS C source file into an object file (*.o*). Because the *s.* of the SCCS files is a prefix, it is incompatible with **make**'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (for example, *.c:*) is the definition of how to build **x** from **x***.c*. In effect, the other suffix is null. This is useful for building targets from only one source file (for example, shell procedures, simple C programs).

Additional suffixes can be given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~ .asm .asm~ .src .src~ .inc .inc~

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus *lib(file.o)* and **$(LIB)***file(.o)* both refer to an archive library that contains *file.o*. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(***file1.o file2.o***)** is not legal. Rules pertaining to archive libraries have the form *.XX.a* where the *XX* is the suffix from which the archive member is to be made.

An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have *lib(file.o)* depend upon *file.o* explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o
```

In fact, the *.c.a* rule listed above is built into **make** and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        $(AR) $(ARFLAGS) lib $?
        rm $?
        @echo lib is now up-to-date

.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object filenames (inside *lib*) whose C source files are out-of-date. The substitution mode translates the *.o* to *.c*. Note also, the disabling of the *.c.a:* rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. Also, you can use the double form of dependency if the archive library contains a mix of assembly programs and C programs: one dependency for C source files and one for assembly source files.

Files

[Mm]akefile and s.[Mm]akefile
/bin/sh

Notes

Lines that begin with a tab character are interpreted as shell command lines. If a line that begins with a tab character does not immediately follow a dependency line, **make** will generate a syntax error.

Some commands return non-zero status inappropriately; use **-i**, **.IGNORE**, or preprend the command with a "-" to overcome the difficulty.

Filenames which include the characters "=", ":", or "@" will not work.

Commands that alter the environment notably **cd**(C), are ineffectual across new-lines in **make**.

The argument list to a command executed by **make** must be less than 5120 characters minus the length of the environment. This is a UNIX limitation.

The syntax (*lib*(*file1.o file2.o file3.o*)) is illegal; you cannot build *lib*(*file.o*) from *file.o*.

Named pipes are not handled well.

See also

cc(CP), **cd**(C), **lex**(CP), **printf**(S), **sccsfile**(FP), **sh**(C), **yacc**(CP)

Standards conformance

make is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

mar

message catalogue archive and library maintainer

Syntax

mar *key* [*option*] *afile names* ...

Description

mar maintains groups of message catalogue files created by **gencat**(CP) combined into a single archive file. Its main use is to create and update library files as used throughout the message catalogue system. Any leading path will be stripped when referencing files that are already part of the archive, hence

```
mar d msgarchive /usr/local/msgfile1
```

is equivalent to

```
mar d msgarchive msgfile1
```

and will delete the file named *msgfile1* from the archive named *msgarchive*.

key is one character from the set **drtpx**, optionally concatenated with **v**. *afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d Delete the named file(s) from the archive file. If no names are given, all files in the archive will be deleted.

r Replace the named files in the archive file or append them to the archive file, if they are not already part of the archive. The archive file will be created if it does not already exist.

t Print a table of contents of the archive file. If no names are given, all files in the archive are printed. If names are given, only those files named will be printed.

p Print the named files to standard output. If no names are given, all files will be printed. When piped to **dumpmsg**(CP) as

```
mar p msgarchive msgfile1 | dumpmsg - -
```

a listing of all the messages in the archive will be printed on standard output.

x Extract the named files. If no names are given, all files in the archive will be extracted. Neither the **x** option nor the **p** option will alter the archive file.

v Verbose. Under the verbose option, **dxtr** give a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the

files, including the name of the file, its last update, its size and the name of the message catalogue as referenced by **catopen**(S), **catgets**(S), and **catgetmsg**(S). The size of the file is identical to the size as reported by **ls**(C).

Environment variables

LC_COLLATE affects the order in which the output is sorted for the **-t** option; and **LC_TIME** affects the contents of date and time strings output by the **-t** and **-v** options. **LANG** provides the necessary defaults if any of these variables are not defined in the environment. If {**LANG**} is not defined, output is sorted in machine collation order, date and time strings default to system specific formats.

Application usage

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

mar will be rather slow on big archives even when only deleting a single file.

Note

This version of **mar** uses a binary format archive which is not portable among the various machines running UNIX. Just as C programs have to be recompiled for each type of machine, message catalogue archives have to be recreated by **mar** as well.

See also

ar(CP), **catopen**(S), **dumpmsg**(CP), **gencat**(CP)

masm

invokes the assembler

Syntax

masm [*options*] *sourcefile*

Description

masm is the Microsoft 8086/286/386 assembler. It reads and assembles 8086/80286/80386 assembly language instructions from *sourcefile* to create a linkable object file *sourcefile.o*, or an executable program *a.out*. The extension *.s* for input filenames is recommended but not required.

masm accepts the following options:

-a Put the assembled output segments in alphabetical order before copying them to the object file.

-c Output cross-reference data for each assembled file to *filename.crf*.

-d Adds a pass 1 listing to the assembly listing *filename.lst*.

-D*sym*[=*value*]
If the optional =*value* is present, the symbol *sym* is defined to be *value*. The value should not contain blanks, commas, or semicolons. If =*value* is not present, *sym* is defined as a null TEXTMACRO.

-e Generates floating point code to emulate the 8087 or 80287 coprocessor. Programs created with this option must be linked with an appropriate math library before being executed.

-h Lists command-line syntax and all valid assembler options.

-I*path* Defines the *path* appended to the **-I** flag as the search path for **#include** files. Up to 10 **#include** *path*s are allowed in one invocation of **masm**.

-l[*listfile*] Creates an assembly listing file with the same base name as the *sourcefile* but with a *.lst* extension. If the *listfile* argument was given, the listing file is called *listfile*. The file lists the source instructions, the assembled (binary) code for each instruction and any assembly errors. If filename is "-," the listing is written to standard output.

-Ml Leave case of symbols alone. This is the default behavior.

-Mu Disable case sensitivity: upper case is treated as being identical to lower case.

-Mx Preserve lower case letters in public and external names only when copying these names to the object file. For all other purposes, **masm** converts lower case to upper case.

-n Suppress symbol tables in the listing files. The **-l** option must also be used for this option to take effect.

-o*objfile* Copies the assembled instructions to the file named *objfile*. This file can be linked to form an executable file only if no errors occurred during the assembly. This option overrides the default object file name.

-p Check for pure code.

-r Do not generate floating point code to emulate 8087 or 80287 coprocessor.

-s Write segments in source-code order (this is the default).

-t Suppress messages for successful assembly.

-v Prints verbose error statistics on console. The default behavior is to display only error counts.

-w[012] Set error-display level, where:

0 No error messages
1 Serious error messages (default)
2 Advisory error messages

-x- Prevent error messages from being written to the screen.

-X Copies to the assembly listing all statements forming the body of an IF directive whose expression (or condition) evaluates to false.

-Zd Put line-number information in the object file.

-Zi Put symbolic and line-number information in the object file.

-z Display source line for each error message (default).

Files

/bin/masm

Notes

The default options are **-Ml** and **-e** which enable case sensitivity and allow emulation of a floating point processor. The following options are flags, with default settings as indicated:

Flag	Default	Meaning of TRUE condition
a	FALSE	Outputs segments alphabetically
c	FALSE	Outputs cross reference data
d	FALSE	Adds pass 1 listing to filename.lst
e	TRUE	Floating Point emulation
n	TRUE	Outputs symbols if -l selected
p	FALSE	Do not check for pure code
r	TRUE	Real 8087 instead of emulated format
s	TRUE	Write segments in source code order
t	TRUE	Suppress error messages for successful assembly
v	FALSE	Prints verbose error statistics
X	FALSE	Toggle setting of conditional flag
z	TRUE	Display source line for each error message

Return value

The **masm** exit codes have the following meanings:

Code	Meaning
0	No error
1	Argument error
2	Unable to open input file
3	Unable to open listing file
4	Unable to open object file
5	Unable to open cross reference file
6	Unable to open include file
7	Assembly errors. If fatal, the object file is deleted.
8	Memory allocation error
9	Real number input not allowed in this version.
10	Unable to define symbol from command line
11	Assembler interrupted

See also

a.out(FP), **cc**(CP), **ld**(CP)
Macro Assembler User's Guide

Standards conformance

masm is a value-added extension of AT&T System V provided by the Santa Cruz Operation.

mcs

manipulate the object file comment section

Syntax

mcs [*options*] *object-file* ...

Description

The **mcs** command is used to add to, delete, print, and compress the contents of the comment section (normally *.comment*) in an object file. The **mcs** command must be given one or more of the options described below. It takes each of the options given and applies them in order to each *object-file*.

If the object file is an archive, the file is treated as a set of individual object files. For example, if the **-a** option is specified, the string is appended to the comment section of each archive element.

The following options are available.

-a *string*
: Append *string* to the comment section of the *object-files*. If *string* contains embedded blanks, it must be enclosed in quotation marks.

-c Compress the contents of the comment section. All duplicate entries are removed. The ordering of the remaining entries is not disturbed.

-d Delete the contents of the comment section from the object file. The object file comment section header is removed also.

-n *name*
: Specify the name of the section to access. By default, **mcs** deals with the section named *.comment*. This option can be used to specify another section.

-p Print the contents of the comment section on standard output. If more than one name is specified, each entry printed is tagged by the name of the file from which it was extracted, using the format "filename:string."

Examples

To print *file*'s comment section:

mcs -p file

To append *string* to *file*'s comment section:

mcs -a string file

Files

TMPDIR/**mcs***	Temporary files.
TMPDIR/*	Temporary files.

TMPDIR is usually */usr/tmp* but can be re-defined by setting the environment variable **TMPDIR** (see **tempnam**() in **tmpnam**(S)).

Notes

This utility operates only on COFF executables.

The **mcs** command cannot add new sections or delete existing sections to executable objects with magic number 0413 (see *a.out*(FP)).

See also

a.out(FP), **cpp**(CP)

Standards conformance

mcs is a valued-added extension of AT&T System V provided by the Santa Cruz Operation.

mkshlib

create a shared library

Syntax

mkshlib -s *specfil* **-t** *target* **[-h** *host***] [-n] [-L** *dir* **...] [-q]**

Description

mkshlib builds both the host and target shared libraries. A shared library is similar in function to a normal, non-shared library, except that programs that link with a shared library will share the library code during execution, whereas programs that link with a non-shared library will get their own copy of each library routine used.

The host shared library is an archive that is used to link-edit user programs with the shared library (see *ar*(FP)). A host shared library can be treated exactly like a non-shared library and should be included on **cc**(CP) command lines in the usual way (see **cc**(CP)). Further, all operations that can be performed on an archive can also be performed on the host shared library.

The target shared library is an executable module that is bound into the user's address space during execution of a program using the shared library. The target shared library contains the code for all the routines in the library and must be fully resolved. The target will be brought into memory during execution of a program using the shared library, and subsequent processes that use the shared library will share the copy of code already in memory. The text of the target is always shared, but each process will get its own copy of the data.

The user interface to **mkshlib** consists of command line options and a shared library specification file. The shared library specification file describes the contents of the shared library.

The **mkshlib** command invokes other tools such as the archiver, **ar**(CP), the assembler, **as**(CP), and the link editor, **ld**(CP). Tools are invoked through the use of **execvp** (see **exec**(S)), which searches directories in the user's **PATH**. Also, prefixes to **mkshlib** are passed in the same manner as prefixes to the **rcc**(CP) command, and invoked tools are given the prefix, where appropriate. For example, **i386mkshlib** will invoke **i386ld**.

mkshlib recognizes the following command line options:

-s *specfil* Specifies the shared library specification file, *specfil*. This file contains the information necessary to build a shared library. Its contents include the branch table specifications for the target, the path name in which the target should be installed, the start addresses of text and data for the target, the initialization specifications for the host, and the list of object files to be included in the shared library (see details below).

-t *target* Specifies the output filename of the target shared library being created. It is assumed that this file will be installed on the target machine at the location given in the specification file (see the **#target** directive below). If the **-n** option is used, then a new target shared library will not be generated.

-h *host* Specifies the output filename of the host shared library being created. If this option is not given, then the host shared library will not be produced.

-n Do not generate a new target shared library. This option is useful when producing only a new host shared library. The **-t** option must still be supplied since a version of the target shared library is needed to build the host shared library.

-L *dir*... Change the algorithm of searching for the host shared libraries specified with the **#objects noload** directive to look in *dir* before looking in the default directories. The **-L** option can be specified multiple times on the command line in which case the directories given with the **-L** options are searched in the order given on the command line before the default directories.

-q Quiet warning messages. This option is useful when warning messages are expected but not desired.

The shared library specification file contains all the information necessary to build both the host and target shared libraries. The contents and format of the specification file are given by the directives listed below.

All directives that can be followed by multi-line specifications are valid until the next directive or the end of the file.

#address *sectname address*
Specifies the start address, *address*, of section *sectname* for the target. This directive typically is used to specify the start addresses of the *.text* and *.data* sections. One **#address** per section name is valid. A **#address** directive must be given exactly once for the *.text* section and once for the *.data* section.

#target ***pathname***

Specifies the absolute path name, ***pathname***, at which the target shared library will be installed on the target machine. The operating system uses this path name to locate the shared library when executing *a.out* files that use this shared library. This directive must be specified exactly once per specification file.

#branch Specifies the start of the branch table specifications. The lines following this directive are taken to be branch table specification lines.

Branch table specification lines have the following format:

funcname <white space> position

where ***funcname*** is the name of the symbol given a branch table entry and ***position*** specifies the position of ***funcname***'s branch table entry. ***position*** may be a single integer or a range of integers of the form ***position1-position2***. Each ***position*** must be greater than or equal to one, the same position can not be specified more than once, and every position, from one to the highest given position, must be accounted for.

If a symbol is given more than one branch table entry by associating a range of positions with the symbol or by specifying the same symbol on more than one branch table specification line, then the symbol is defined to have the address of the highest associated branch table entry. All other branch table entries for the symbol can be thought of as "empty" slots and can be replaced by new entries in future versions of the shared library. Only functions should be given branch table entries, and those functions must be *external* symbols.

This directive must be specified exactly once per shared library specification file.

#objects The lines following this directive are taken to be the list of input object files in the order they are to be loaded into the target. The list simply consists of each path name followed by a newline character. This list is also used to determine the input object files for the host shared library, but the order for the host is given by running the list through **lorder**(CP) and **tsort**(CP).

This directive must be specified exactly once per shared library specification file.

#objects noload

The **#objects noload** is followed by a list of host shared libraries. These libraries are searched in the order listed to resolve undefined symbols from the library being built. During the search it is considered an error if a non-shared version of a symbol is found before a shared version of the symbol.

Each name given is assumed to be a path name to a host or an argument of the form **-l***X*, where *libX.a* is the name of a file in *LIBDIR* or *LLIBDIR*. This behavior is identical to that of **ld**, and the **-L** option can be used on the command line to specify other directories in which to locate these archives.

Note that if a host shared library is specified using **#objects noload**, any **cc** command that links to the shared library being built will need to specify that host also.

#hide linker [*]

This directive changes symbols that are normally *external* into *static* symbols, local to the library being created. A regular expression may be given (see **sh**(C), **find**(C)), in which case all *external* symbols matching the regular expression are hidden; the **#export** directive (see below) can be used to counter this effect for specified symbols.

The optional * is equivalent to the directive

```
#hide linker
  *
```

and causes all *external* symbols to be made into *static* symbols.

All symbols specified in **#init** and **#branch** directives are assumed to be *external* symbols, and cannot be changed into *static* symbols using the **#hide** directive.

#export linker [*]

Symbols given in the **#export** directive are *external* symbols (global among files) that, because of a regular expression in a **#hide** directive, would otherwise have been made *static*. For example,

```
#hide linker *
#export linker
  one
  two
```

causes all symbols except *one*, *two*, and those used in **#branch** and **#init** entries to be tagged as *static*.

#init *object*

Specifies that the object file, *object*, requires initialization code. The lines following this directive are taken to be initialization specification lines.

Initialization specification lines have the following format:

ptr <white space> import

ptr is a pointer to the associated imported symbol, ***import***, and must be defined in the current specified object file, ***object***. The initialization code generated for each such line is of the form:

ptr = &import;

All initializations for a particular object file must be given once and multiple specifications of the same object file are not allowed.

#ident ***string***
Specifies a string, ***string***, to be included in the *.comment* section of the target shared library.

Specifies a comment. All information on the line beginning with **##** is ignored.

Files

TEMPDIR/* Temporary files.

TEMPDIR is usually */usr/tmp* but can be re-defined by setting the environment variable **TMPDIR** (see **tempnam()** in **tmpnam**(S)).

LIBDIR Usually */lib*.

LLIBDIR Usually */usr/lib*.

Notes

The **-n** option cannot be used with the **#objects noload** directive.

If **mkshlib** is asked to create a host library and a host of that name already exists, **mkshlib** will update the host using **ar -ru**. This means that you should always remove the host before rebuilding whenever an object file previously included in the library is removed or renamed.

If the address specified with the **#address** directive is outside user space, the library build may look successful, but if you try to use it, it might not work.

See also

a.out(FP), **ar**(CP), **ar**(FP), **as**(CP), **cc**(CP), **chkshlib**(CP), **ld**(CP), **lorder**(CP), **rcc**(CP), **tsort**(CP)

mkstr

creates an error message file from C source

Syntax

mkstr [-] *messagefile prefix file...*

Description

mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

mkstr will process each specified *file*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. The optional dash ("-") causes the error messages to be placed at the end of the specified message file for recompiling part of a large **mkstred** program.

A typical **mkstr** command line is:

```
mkstr pistrings xx *.c
```

This command causes all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx*.

To process the error messages in the source to the message file, **mkstr** keys on the string 'error("' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a newline character; the null character terminates the message so it can be easily used when retrieved, the newline character makes it possible to sensibly **cat** the error message file to see its contents. The massaged copy of the input file then contains a **lseek** pointer into the file which can be used to retrieve the message. For example, the command changes

```
error("Error on reading", a2, a3, a4);
```

into

```
error(m, a2, a3, a4);
```

where *m* is the seek position of the string in the resulting error message file. The programmer must create a routine **error** which opens the message file, reads the string, and prints it out. The following example illustrates such a routine.

```
char    efilname[] =  "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
int a1, a2, a3, a4;
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) ||
                  read(efil, buf, 256) <= 0) {
                printf("Unable to find error msg ");
                printf("at seek address %d0,a1);
                exit(1);
                }
        printf(buf, a2, a3, a4);
}
```

Notes

All the arguments except the name of the file to be processed are optional.

See also

lseek(S), **xstr**(CP)

Standards conformance

This utility was developed at the University of California at Berkeley and is used with permission.

nm

print name list of common object file

Syntax

nm [-oxhvnefurpVT] *filename...*

Description

The **nm** command displays the symbol table of each common object file, *filename*. *filename* may be a relocatable or absolute common object file, or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

Name The name of the symbol.

Value Its value expressed as an offset or an address depending on its storage class.

Class Its storage class.

Type Its type and derived type. If the symbol is an instance of a structure or of a union, then the structure or union tag will be given following the type (for example, struct-tag). If the symbol is an array, then the array dimensions will be given following the type (for example, char[n][m]). Note that the object file must have been compiled with the **-g** option of the **cc**(CP) command for this information to appear.

Size Its size in bytes, if available. Note that the object file must have been compiled with the **-g** option of the **cc**(CP) command for this information to appear.

Line The source line number at which it is defined, if available. Note that the object file must have been compiled with the **-g** option of the **cc**(CP) command for this information to appear.

Section For storage classes static and external, the object file section containing the symbol (for example, text, data, or bss).

The output of **nm** may be controlled using the following options:

-o Print the value and size of a symbol in octal instead of decimal.

-x Print the value and size of a symbol in hexadecimal instead of decimal.

-h Do not display the output header data.

-v Sort external symbols by value before they are printed.

-n Sort external symbols by name before they are printed.

-e Print only external and static symbols.

-f Produce full output. Print redundant symbols (*.text*, *.data*, *.lib*, and *.bss*), normally suppressed.

-u Print undefined symbols only.

-r Prepend the name of the object file or archive to each output line.

-p Produce easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **B** (.bss symbol), **T** (text segment symbol), **D** (data segment symbol), **S** (user-defined segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external), the type letter is in lowercase.

-V Print the version of the **nm** command executing on the standard error output.

-T By default, **nm** prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **-T** option causes **nm** to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm** *name* **-e -v** and **nm -ve** *name* print the static and external symbols in *name*, with external symbols sorted by value.

Files

TMPDIR/* Temporary files.

TMPDIR is usually */usr/tmp* but can be re-defined by setting the environment variable **TMPDIR** (see **tempnam()** in **tmpnam**(S)).

Diagnostics

"nm: name: cannot open"
name cannot be read.

"nm: name: bad magic"
name is not a common object file.

"nm: name: no symbols"
The symbols have been stripped from *name*.

Notes

If you are using XENIX binaries, refer to the XENIX manual page for this utility for information on the appropriate usage with XENIX binaries.

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **-v** and **-n** options should be used only in conjunction with the **-e** option.

The **-n** option generates an error with long name lists. To sort a long name list, instead of using the **-n** option, use the command line:

```
nm executable | sort
```

This utility determines whether an object file is in OMF or COFF format and executes the appropriate binaries in each case. Since this utility relies on the files in */lib/coff* and */lib/xout*, changing or moving any of those files results in an error.

See also

a.out(FP), **ar**(FP), **as**(CP), **cc**(CP), **ld**(CP), **tmpnam**(S)

Standards conformance

nm is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

os2ld

OS/2 cross linker

Syntax

os2ld [*options*] *file* ...

Description

os2ld links the object *file*(s) to create a program for execution under OS/2. Although similar to **ld**(CP), some **os2ld** options differ significantly from **ld**. The options are described below:

-B Produce an executable *.COM* file.

-C Treat upper and lower case characters in symbol names as identical.

-D Perform DS allocation, which is generally used in conjunction with the **-H** option.

-F *num*
Set stack size to the hexadecimal number *num*. **os2ld** will set the stack segment to *num* bytes in the output file.

-H Load high. This option sets a field in the header of the executable file to tell OS/2 to load the program at the highest available position in memory. It is most often used with programs in which data precedes code in the memory image.

-L Include line numbers in the listing file (if any). **os2ld** cannot put line numbers in the listing file if the source translator hasn't put them in the object file.

-M
Include public symbols in the list file. The symbols are sorted twice, lexicographically and by address.

-m *filename*
Create a map file *filename* with information about the segments and groups in the executable. Public symbols and line numbers will be listed in this file if the **-M** and **-L** options are given.

-n *num*
Set name length to the decimal number *num*. **os2ld** truncates all public and external symbols longer than *num* characters.

-o *filename*
Name output file to *filename* (the default name is *a.out*).

-P Pack code segments.

-Pc[<*dec#*>]
Pack code segments; the default limit for a segment is 64k minus 36.

-Pd[<*dec#*>]
Pack data segments; default limit is 64k.

-Q Don't pack code segments; this option is the default for segmented-executable pack code.

-S *num*
Set segment limit to *num*, a decimal number between 1 and 1024 (the default is 128). *num* limits the number of different segments that may be linked together. The higher the value given, the slower the link will be.

-u *name*
Name undefined symbol *name*. **os2ld** will enter the given name into its symbol table as an undefined symbol. The **-u** option may appear more than once on the command line.

As with **ld**, the files passed to **os2ld** may be either UNIX-style libraries (objects collected using **ar**(CP) or ordinary 8086 object files. Unless the **-u** option appears, at least one of the files passed to **os2ld** must be an ordinary object file. Libraries are searched only after all the ordinary object files have been processed.

Files

/usr/bin/os2ld

See also

ar(CP), **as**(CP), **cc**(CP), **dosld**(CP), **ld**(CP)

Standards conformance

os2ld is a value-added extension of AT&T System V provided by the Santa Cruz Operation.

prof

display profile data

Syntax

prof [-tcan] [-ox] [-g] [-h] [-m *mdata*] [-s] [-z] [*prog*]

Description

The **prof** command interprets a profile file produced by the **monitor**(S) function. The symbol table in the object file *prog* (called *a.out* by default) is read and correlated with a profile file (*mon.out* by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **-t**, **-c**, **-a**, and **-n** determine the type of sorting of the output lines:

-t Sort by decreasing percentage of total time (default).

-c Sort by decreasing number of calls.

-a Sort by increasing symbol address.

-n Sort lexically by symbol name.

The mutually exclusive options **-o** and **-x** specify the printing of the address of each symbol monitored:

-o Print each symbol address (in octal) along with the symbol name.

-x Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

-g Include non-global symbols (static functions).

-h Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)

-m *mdata*
Use the file *mdata* instead of *mon.out* as the input profile.

-s Print a summary of several of the monitoring parameters and statistics on standard error.

-z Include all symbols in the profile range (see **monitor**(S)), even if associated with zero number of calls and zero time.

A program creates a profile if it has been loaded with the **-p** option of **cc**(CP). This option to **cc** arranges for calls to **monitor**(S) at the beginning and end of execution. It is the call to **monitor** at the end of execution that causes the profile to be written. The number of calls to a function is tallied if the **-p** option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable **PROFDIR**. If **PROFDIR** does not exist, *mon.out* is produced in the directory that is current when the program terminates. If **PROFDIR** = *string*, "string/pid.progname" is produced, where *progname* consists of argv[0] with any path prefix removed, and *pid* is the program's process id. If **PROFDIR** is the null string, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the **MARK** macro (see **prof**(M)).

Files

mon.out Profile.

a.out Namelist.

Warning

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely.

The times for static functions are attributed to the preceding external text symbol if the **-g** option is not used. However, the call counts for the preceding function are still correct (that is, the static function call counts are not added in with the call counts of the external function).

Notes

If you are using XENIX binaries, please refer to the XENIX manual page for this utility for information on the appropriate usage with XENIX binaries.

Only programs that call **exit**(S) or return from **main** will cause a profile file to be produced, unless a final call to monitor is explicitly coded.

The use of the **-p** option to **cc**(CP) to invoke profiling imposes a default limit of 600 (300 on the PDP-11) functions that may have call counters established during program execution. If this limit is exceeded, other data will be overwritten and the *mon.out* file will be corrupted. For more counters you can call **monitor**(S) directly. Alternatively, the value of the global variable **_nfunc**, located in *libc.a*, can be increased before compiling the code with the **-p** option. The number of call counters used will be reported automatically by **prof** whenever the number exceeds 5/6 of the maximum.

See also

cc(CP), **exit**(S), **monitor**(S), **prof**(M), **profil**(S)

Standards conformance

prof is conformant with:

AT&T SVID Issue 2.

prs

print an SCCS file

Syntax

prs [**-d**[*dataspec*]] [**-r**[*SID*]] [**-e**] [**-l**] [**-c**[*date-time*]] [**-a**] *files*
prs -

Description

The **prs** command prints, on standard output, parts or all of an SCCS file (see *sccsfile*(FP)) in a user-supplied format. If a directory is named, **prs** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of "-" is given, the standard input is read; each line of standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to **prs** may appear in any order. All the described arguments apply independently to each named file:

-d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see "Data Keywords") interspersed with optional user-supplied text.

-r[*SID*] Used to specify the SCCS identification (*SID*) string of a delta for which information is desired. If no *SID* is specified, the *SID* of the most recently created delta is assumed.

-e Requests information for all deltas created *earlier* than and including the delta designated via the **-r** option or the date given by the **-c** option.

-l Requests information for all deltas created *later* than and including the delta designated via the **-r** option or the date given by the **-c** option.

-c[*date-time*] The cutoff [*date-time*] is in the form:

YY[MM[DD[HH[MM[SS]]]]]

Units omitted from the [*date-time*] default to their maximum possible values; that is, **-c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters may separate the various two-digit pieces of the cutoff date in the form: **-c77/2/2 9:22:25**. The **-c** option must be used with the **-e** or **-l** option.

-a Requests printing of information for both removed (delta type **R**) and existing (delta type **D**) deltas (see **rmdel**(CP). If **-a** is not specified, information is provided for existing deltas only.

Data keywords

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(FP)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a ***dataspec***.

The information printed by **prs** consists of the user-supplied text and appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the ***dataspec***. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multiline* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords. A tab is specified by *t* and new-line is specified by *n*. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	D~or~R	S
:I:	SCCS ID string (SID)	"	:R::L::B::S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S

(Continued on next page)

(Continued)

TABLE 1. SCCS Files Data Keywords

Keyword	Data Item	File Section	Value	Format
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS:~:DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS:~:DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS:~:DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	yes~or~no	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	yes~or~no	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	yes~or~no	S
:J:	Joint edit flag	"	yes~or~no	S
:LK:	Locked releases	"	:R:...	S
:Q:	User-defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	yes~or~no	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of what(CP) string	N/A	:Z::M:\t:I:	S
:A:	A form of what(CP) string	N/A	:Z::Y:~:M:~:I::Z:	S
:Z:	what(C) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt:~=~:DT:~:I:~:D:~:T:~:P:~:DS:~:DP:

Examples

prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:
xyz
131
abc

prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a special case: prs *s.file* may produce on the standard output for each delta table entry of the "D" type. For example:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta

The only option allowed to be used with the special case is **-a**.

Files

/tmp/pr????? Temporary files.

See also

admin(CP), **delta**(CP), **get**(CP), **sccsfile**(FP)

Standards conformance

prs is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

rcc

AT&T C compiler

Syntax

rcc [*options*] *files*

Description

The **rcc** command invokes the AT&T C compiler. Note that this is a different compiler from the Microsoft C compiler that is available through the **cc**(CP) command.

The AT&T compilation tools consist of a preprocessor, compiler, optimizer, assembler, and link editor. The **rcc** command processes the supplied options and then executes the various tools with the proper arguments. The **rcc** command accepts several types of files as arguments.

Files whose names end with *.c* are taken to be C source programs and may be preprocessed, compiled, optimized, assembled, and linked. Compilation may be stopped after the completion of any pass if appropriate options are supplied. If compilation runs through the assembler, then an object file is produced called *source.o*, with *.o* substituted for *.c*. The *.o* file is normally deleted if a single C program is compiled and then immediately linked. In the same way, files whose names end in *.s* are taken to be assembly source programs and may be assembled and linked; files whose names end in *.i* are taken to be preprocessed C source programs and may be compiled, optimized, assembled, and linked. Files whose names do not end in *.c*, *.s*, or *.i* are passed to the linker.

Since **rcc** usually creates files in the current directory during compilation, you must run **rcc** in a directory in which a file can be created.

The following options are accepted by **rcc**:

-c Suppress the linking phase of compilation and do not remove object files.

-dl Do not generate symbolic debugging line-number information. The **-dl** flag may be used with the **-ds** flag as **-dsl** (which is the default unless the **-g** flag is given).

-ds Do not generate symbol attribute information for the symbolic debugger.

-g Generate additional information needed for debuggers like **dbxtra**(CP), **codeview**(CP), and **sdb**(CP).

-o *outfile*
Produce an output object, *outfile* (the default name is *a.out*). This is a link editor option.

-p Produce code that counts the number of times each routine is called; also, if linking takes place, link profiled versions of *libc.a* and *libm.a* (with **-lm**) and call **monitor**(S) automatically. *mon.out* will be created on normal termination of the object program. An execution profile can be generated by **prof** (see the **prof**(CP) manual page for more information).

-ql Provides line profiling, that is, "coverage" and "count" statistics for each line of the program. The output from line profiling can be examined with **lprof**(CP).

-qp Produce profiled code where the **p** argument produces identical results to the **-p** option (allows profiling with **prof**(CP)).

-E Run only **cpp**(CP) on the named C programs, and send the result to standard output.

-H Print out (on standard error) the path name of each file included during the current compilation.

-O Do compilation phase optimization. This option has no effect on *.s* files.

-P Run only **cpp**(CP) on the named C source files and leave the results in corresponding files suffixed *.i*. This option is passed to **cpp**(CP).

-S Compile but do not assemble the named C source files, and leave the assembly-language output in corresponding files suffixed *.s*.

-V Print the version(s) of the compiler, optimizer, assembler and/or linker invoked.

-Wc,*arg1*[,*arg2*...]
Hand off the argument[s] ***argi*** to pass ***c*** where ***c*** is one of [***p02al***] indicating the preprocessor, compiler, optimizer, assembler, or link editor, respectively. For example: **-Wa,*-m*** passes ***-m*** to the assembler.

-Y [*p02alSILU*],*dirname*
Specify a new path name, ***dirname***, for the locations of the tools and directories designated in the first argument. [***p02alSILU***] represents:

p preprocessor

0 compiler

2 optimizer

a assembler

l link editor

S directory containing the start-up routines

I default include directory searched by **cpp**(CP)

L first default library directory searched by **ld**(CP)

U second default library directory searched by **ld**(CP)

If the location of a tool is being specified, then the new path name for the tool will be ***dirname/tool***. If more than one **-Y** option is applied to any one tool or directory, then the last occurrence holds.

-Zp[*1*|*2*|*4*]
Packs structure members in memory. Normally, structure members are aligned as follows: items of type ***char*** are byte-aligned, items of type ***short*** are aligned on two-byte boundaries, and all other types of structure members are word-aligned.

Specifying an option to **-Zp** will force alignment on the given byte boundary. If no option is used with **-Zp**, structure members will be packed on one-byte boundaries. The alignment may be altered with the **#pragma pack** preprocessor directive.

The **rcc** command also recognizes **-C**, **-D**, **-I**, and **-U** and passes these options and their arguments directly to the preprocessor without using the **-W** option. Similarly, the **rcc** command recognizes **-a**, **-l**, **-m**, **-r**, **-s**, **-t**, **-u**, **-x**, **-z**, **-L**, **-M**, and **-V** and passes these options and their arguments directly to the loader. See the **cpp**(CP) and **ld**(CP) manual pages for descriptions.

Other arguments are taken to be C compatible object programs or libraries of C compatible routines, and are passed directly to the link editor. These programs, together with the results of any compilations specified, are link edited (in the order given) to produce an executable program with name *a.out* unless the **-o** option of the link editor is used.

If the **rcc** command is put in a file *prefix*cc the prefix will be parsed off the command and used to call the tools, for example, *prefix*tool. For example, *OLD*rcc will call *OLD*cpp, *OLD*comp, *OLD*optim, *OLD*as, and *OLD*ld and will link *OLD*crt1.o. Therefore, you must be careful when moving the **rcc** command around. The prefix will apply to the preprocessor, compiler, optimizer, assembler, link editor, and the start-up routines.

The C language standard was extended to allow arbitrary length variable names. The option pair "**-Wp,-T -W0,-XT**" will cause **rcc** to truncate arbitrary length variable names.

Files

file.c	C source file.
file.i	Preprocessed C source file.
file.o	Object file.
file.s	Assembly language file.
a.out	Link-edited output.
LIBDIR/*rcrt1.o	Start-up routine.
LIBDIR/rcrtn.o	Start-up routine.
TMPDIR/*	Temporary files.
LIBDIR/rcpp	Preprocessor, **cpp**(CP).
LIBDIR/rcomp	Compiler.
LIBDIR/roptim	Optimizer.
BINDIR/as	Assembler, **as**(CP).
BINDIR/ld	Link editor, **ld**(CP).
LIBDIR/	Standard C library.
LIBDIR/libc_s.a	Standard C shared library.

LIBDIR is usually */lib*.

BINDIR is usually */bin*.

TMPDIR is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**( ) in **tmpnam**(S)).

Notes

By default, the return value from a compiled C program is completely random. The only ways to guarantee a specific return value are to call **exit**(S) explicitly or to exit the function **main**( ) with a *"return expression;"* construct.

See also

as(CP), **cpp**(CP), **gencc**(CP), **ld**(CP), **lint**(CP), **prof**(CP), **sdb**(CP), **tmpnam**(S)

rcflow

generate C flowgraph

Syntax

rcflow [-r] [-ix] [-i_] [-d*num*] ***files***

Description

The **rcflow** command analyzes a collection of C, **yacc**, **lex**, assembler, and object files and attempts to build a graph charting the external references. Files suffixed with *.y*, *.l*, and *.c* are "yacc'ed," "lex'ed," and C-preprocessed as appropriate. The results of the preprocessed files, and files suffixed with *.i*, are then run through the first pass of **rlint**(CP). Files suffixed with *.s* are assembled. Assembled files, and files suffixed with *.o*, have information extracted from their symbol tables. The results are collected and turned into a graph of external references which is sent to standard output.

Each line of output begins with a reference number, followed by a suitable number of tabs indicating the level, then the name of the global symbol followed by a colon and its definition. Normally only function names that do not begin with an underscore are listed (see the **-ix** and **-i_** options below). For information extracted from C source, the definition consists of an abstract type declaration (for example, ***char** ******), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (for example, ***text***). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only "<>" is printed.

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}

f()
{
        i = h();
}
```

The command

```
rcflow -ix file.c
```

produces the output

```
1        main: int(), <file.c 4>
2                f: int(), <file.c 11>
3                        h: <>
4                        i: int, <file.c 1>
5                g: <>
```

When the nesting level becomes too deep, the output of **rcflow** can be piped to **pr**(C), using the **-e** option, to compress the tab expansion to something less than every eight spaces.

In addition to the **-D**, **-I**, and **-U** options (which are interpreted just as they are by **cc**(CP) and **cpp**(CP)), the following options are interpreted by **rcflow**:

-r Reverse the "caller:callee" relationship, producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

-ix Include external and static data symbols. The default is to include only functions in the flowgraph.

-i_ Include names that begin with an underscore. The default is to exclude these functions (and data if **-ix** is used).

-d*num*
The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a non-positive integer will be ignored.

Diagnostics

rcflow complains about bad options. It also complains about multiple definitions and only believes the first one. Other messages may come from the various programs used (for example, the C-preprocessor).

Notes

Files produced by **lex**(CP) and **yacc**(CP) cause the re-ordering of line number declarations which can confuse **rcflow**. To get proper results, feed **rcflow** the **yacc** or **lex** input.

See also

as(CP), **cc**(CP), **cpp**(CP), **lex**(CP), **nm**(CP), **pr**(C), **rlint**(CP), **yacc**(CP)

Standards conformance

rcflow is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

rcxref

generate C program cross-reference

Syntax

rcxref [*options*] *files*

Description

The **rcxref** command analyzes a collection of C files and attempts to build a cross-reference table. The **rcxref** command uses a special version of **cpp** to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or, with the **-c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **-D**, **-I**, and **-U** options (which are interpreted just as they are by **rcc**(CP) and **cpp**(CP.)

The following *options* are interpreted by **rcxref**:

-c Print a combined cross-reference of all input files.

-w *<num>*
Width option which formats output no wider than *<num>* (decimal) columns. This option will default to 80 if *<num>* is not specified or is less than 51.

-D*name=def*
Define *name* with value *def* as if by a **#define**. If no *=def* is given, *name* is defined with value 1.

-o *file*
Direct output to *file*.

-s Operate silently; do not print input file names.

-t Format listing for 80-column width.

Files

LLIBDIR	Usually */usr/lib*.
LLIBDIR/xcpp	Special version of the C preprocessor.

Diagnostics

Error messages are unusually cryptic, but usually mean that you cannot compile these files.

Notes

The **rcxref** command considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program that **#includes** *ctype.h*, will contain many declarations of the variable **c**.

See also

rcc(CP), **cpp**(CP)

Standards conformance

rcxref is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

regcmp

regular expression compile

Syntax

regcmp [-] *files*

Description

The **regcmp** command performs a function similar to **regcmp**(S) and, in most cases, precludes the need to call **regcmp**(S) from C programs. This saves on both execution time and program size. The command **regcmp** compiles the regular expressions in *file* and places the output in *file.i*. If the - option is used, the output will be placed in *file.c*. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of **regcmp** is C source code. Compiled regular expressions are represented as *extern char* vectors. *file.i* files may thus be *included* in C programs, or *file.c* files may be compiled and later loaded. In the C program which uses the **regcmp** output, **regex** (*abc*, *line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

Examples

```
name  "([A-Za-z][A-Za-z0-9_]*)$0"

telno  "\({0,1}([2-9][01][1-9])$0\){0,1} *"
       "([2-9][0-9]{2})$1[ -]{0,1}"
       "([0-9]{4})$2"
```

In the C program that uses the **regcmp** output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

See also

regcmp(S)

rlint

a C program checker

Syntax

rlint [*option*] ... *file* ...

Description

The **rlint** command attempts to detect features of C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

rlint is based on the AT&T compiler **rcc**, and does not accept either the ANSI or Microsoft dialects of the C language. For checking programs written with either of these dialects, use the **lint** command (see the **lint**(CP) manual page).

Arguments whose names end with *.c* are taken to be C source files. Arguments whose names end with *.rln* are taken to be the result of an earlier invocation of **rlint** with either the **-c** or the **-o** option used. The *.rln* files are analogous to *.o* (object) files that are produced by the **rcc**(CP) command when given a *.c* file as input. Files with other suffixes are warned about and ignored.

The **rlint** command will take all the *.c*, *.rln*, and *llib-lx.rln* (specified by **-lx**) files and process them in their command line order. By default, **rlint** appends the standard C rlint library (*llib-lc.rln*) to the end of the list of files. However, if the **-p** option is used, the portable C rlint library (*llib-port.rln*) is appended instead. When the **-c** option is not used, the second pass of **rlint** checks this list of files for mutual compatibility. When the **-c** option is used, the *.rln* and the *llib-lx.rln* files are ignored.

Any number of **rlint** options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

-a Suppress complaints about assignments of long values to variables that are not long.

-b Suppress complaints about break statements that cannot be reached. (Programs produced by **lex** or **yacc** will often result in many such complaints.)

-h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

-u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running **rlint** on a subset of files of a larger program.)

-v Suppress complaints about unused arguments in functions.

-x Do not report variables referred to by external declarations but never used.

The following arguments alter **rlint**'s behavior:

-c Cause **rlint** to produce a *.rln* file for every *.c* file on the command line. These *.rln* files are the product of **rlint**'s first pass only, and are not checked for inter-function compatibility.

-l*x* Include additional rlint library *llib-lx.rln*. For example, you can include a rlint version of the math library *llib-lm.rln* by inserting **-lm** on the command line. This argument does not suppress the default use of *llib-lc.rln*. These rlint libraries must be in the assumed directory. This option can be used to reference local rlint libraries and is useful in the development of multifile projects.

-n Do not check compatibility against either the standard or the portable rlint library.

-o *lib*
Cause **rlint** to create a rlint library with the name *llib-llib.rln*. The **-c** option nullifies any use of the **-o** option. The rlint library produced is the input that is given to **rlint**'s second pass. The **-o** option simply causes this file to be saved in the named rlint library. To produce a *llib-llib.rln* without extraneous messages, use of the **-x** option is suggested. The **-v** option is useful if the source file(s) for the rlint library are just external interfaces (for example, the way the file *llib-lc* is written). These option settings are also available through the use of "rlint comments" (see below).

-p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

The **-D**, **-U**, and **-I** options of **cpp**(CP) and the **-g** and **-O** options of **rcc**(CP) are also recognized as separate arguments. The **-g** and **-O** options are ignored, but, by recognizing these options, **rlint**'s behavior is closer to that of the **rcc**(CP) command. Other options are warned about and ignored. The preprocessor symbols "lint" and "__lint" are defined to allow certain questionable code to be altered or removed for **rlint**. Therefore, the symbols "lint" and "__lint" should be thought of as reserved words for all code that you plan to check with **rlint**.

Certain conventional comments in the C source will change the behavior of **rlint**:

/*ARGSUSED*/
Turns on the **-v** option for the next function.

/*LINTLIBRARY*/
Placed at the beginning of a file turns off complaints about unused functions and function arguments in this file. This is equivalent to using the **-v** and **-x** options.

/*NOTREACHED*/
Placed at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like **exit**(S).)

/*VARARGS*n*/
Suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

The **rlint** command produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the **-c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the **-c** and the **-o** options allows for incremental use of **rlint** on a set of C source files. Generally, one invokes **rlint** once for each source file with the **-c** option. Each of these invocations produces a *.rln* file for each *.c* file, and prints all messages that are about just that source file. After all the source files have been separately run through **rlint**, it is invoked once more (without the **-c** option), listing all the *.rln* files with the needed **-lx** options. This will print all the interfile inconsistencies. This scheme works well with **make**(CP); it allows **make** to be used to **rlint** only the source files that have been modified since the last time the set of source files were **rlint**ed.

Files

LLIBDIR	the directory where the rlint libraries specified by the **-lx** option must exist, usually */usr/lib*.
LLIBDIR/**rlint**[12]	first and second passes
LLIBDIR/llib-lc.rln	declarations for C Library functions (binary format; source is in *LLIBDIR/llib-lc.c*)

LLIBDIR/llib-port.rln	declarations for portable functions (binary format; source is in *LLIBDIR/llib-port.c*)
LLIBDIR/llib-lm.rln	declarations for Math Library functions (binary format; source is in *LLIBDIR/llib-lm.c*)
TMPDIR/*rlint*	temporaries
TMPDIR	usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see **tempnam**() in **tmpnam**(S)).

Notes

exit(S), **setjmp**(S), and other functions that do not return are not understood; this causes various mistaken diagnoses.

See also

cpp(CP), **make**(CP), **rcc**(CP)

Standards conformance

rlint is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

rmdel

remove a delta from an SCCS file

Syntax

rmdel -r*SID files*
rmdel -

Description

The **rmdel** command removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the delta specified must *not* be that of a version being edited for the purpose of making a delta (that is, if a *p-file* (see **get**(CP)) exists for the named SCCS file, the delta specified must *not* appear in any entry of the *p-file*).

The **-r** option is used for specifying the *SID* (SCCS identification) level of the delta to be removed.

If a directory is named, **rmdel** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of "-" is given, standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed.

Simply stated, if you make a delta then you can remove it, or if you own the file and directory then you can remove a delta.

Files

x.file (see **delta**(CP))

z.file (see **delta**(CP))

See also

delta(CP), **get**(CP), **prs**(CP), **sccsfile**(FP)

Standards conformance

rmdel is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

sact

print current SCCS file editing activity

Syntax

sact *files*
sact -

Description

The **sact** command informs the user of any impending deltas to a named SCCS file. This situation occurs when **get**(CP) has been executed with the **-e** option without a subsequent execution of **delta**(CP). If a directory is named on the command line, **sact** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of "-" is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1 Specifies the *SID* of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2 Specifies the *SID* for the new delta to be created.

Field 3 Contains the logname of the user who will make the delta (for example, executed a **get** for editing).

Field 4 Contains the date that **get -e** was executed.

Field 5 Contains the time that **get -e** was executed.

See also

delta(CP), **get**(CP), **unget**(CP)

Standards conformance

sact is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

sccsdiff

compare two versions of an SCCS file

Syntax

sccsdiff -r*SID1* **-r***SID2* [-p] [-s*n*] *files*

Description

The **sccsdiff** command compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

-r*SID?* *SID1* and *SID2* the deltas of an SCCS file that are to be compared. Versions are passed to **bdiff**(C) in the order given.

-p Pipe output for each file through **pr**(C).

-s*n* *n* is the file segment size that **bdiff** will pass to **diff**(C). This is useful when **diff** fails due to a high system load.

Diagnostics

"file: No differences" If the two versions are the same.

See also

bdiff(C), **get**(CP), **pr**(C)

sdb

symbolic debugger

Syntax

sdb [-w] [-W] [[*objfile* [*corefile*]] [*directory*:*directory*:...] [[*corefile* | -]]

Description

The **sdb** command calls a symbolic debugger that can be used to debug C programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

The command-line options available are:

-w While a process is running under **sdb**, all addresses refer to the executing program; otherwise they refer to *objfile* or *corefile*. An initial argument of **-w** permits overwriting locations in *objfile*.

-W By default, warnings are provided if the source files used in producing *objfile* cannot be found, or are newer than *objfile*. This checking feature and the accompanying warnings may be disabled by the use of the **-W** flag.

objfile is a COFF format executable program file which has been compiled with the **-g** or **-Zi** (debug) option. If it has not been compiled with a debug option specified, the symbolic capabilities of **sdb** will be limited, but the file can still be examined and the program debugged. The default for *objfile* is *a.out*. *corefile* is assumed to be a core image file produced after executing *objfile*; the default for *corefile* is *core*. The core file need not be present. A "-" in place of *corefile* will force **sdb** to ignore any core image file.

It is useful to know that at any time there is a *current line* and *current file*. If *corefile* exists, then they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in **main**( ). The current line and file may be changed with the source file examination commands.

Names of variables are written just as they are in C. **sdb** does not truncate names. Variables local to a procedure may be accessed using the form *procedure*:*variable*. If no procedure name is given, the procedure containing the current line is used by default.

It is also possible to refer to structure members as *variable* pointers to structure members as *variable*->*member*, and array elements as *variable*[*number*]. Pointers may be dereferenced by using the form *pointer*[0]. Combinations of these forms may also be used. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the

structure, and the template used for the structure is that of the last structure referenced by **sdb**. An unqualified structure variable may also be used with various commands. Generally, **sdb** will interpret a structure as a set of variables. Thus, **sdb** will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value **sdb** displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable*[*number*][*number*]..., or as *variable* [*number,number,*...]. In place of *number*, the form *number;number* may be used to indicate a range of values, an asterisk (*) may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, **sdb** displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal, or hexadecimal.

Line numbers in the source program are referred to as *filename:number* or *procedure:number*. In either case the number is relative to the beginning of the file. If no *procedure* or *filename* is given, the current file is used by default. If no *number* is given, the first line of the named procedure or file is used.

Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*) and the *file address* corresponding to a written *address* is calculated as follows:

b1≤address<e1
then
file address=address+f1-b1

otherwise

b2≤address<e2
then
file address=address+f2-b2

otherwise, the requested *address* is not legal. In some cases (for example, programs with separated text and data space) the two segments for a file may overlap.

The initial setting of both mappings is suitable for normal *a.out* and *core* files. If either file is not of the kind expected, then, for that file, *b1* is set to 0, *e1* is set to the maximum file size, and *f1* is set to 0; in this way the whole file can be examined with no address translation.

In order for **sdb** to be used on large files, all appropriate values are kept as signed 32-bit integers.

Commands

The commands for examining data in the program are:

t Print a stack trace of the terminated or halted program.

T Print the top line of the stack trace.

variable/clm

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

- **b** one byte
- **h** two bytes (half word)
- **l** four bytes (long word)

Legal values for *m* are:

- **c** character
- **d** decimal
- **u** unsigned decimal
- **o** octal
- **x** hexadecimal
- **f** 32-bit single precision floating point
- **g** 64-bit double precision floating point
- **s** Assume *variable* is a string pointer and print the characters starting at the address pointed to by the variable.
- **a** Print characters starting at the variable's address. This format may not be used with register variables.
- **p** Pointer to procedure.
- **i** Disassemble machine-language instruction with addresses printed numerically and symbolically.
- **I** Disassemble machine-language instruction with addresses just printed numerically.

Length specifiers are only effective with the **c**, **d**, **u**, **o**, and **x** formats. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, **sdb** chooses a length and a format suitable for the variable's type as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells **sdb** to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the *variable*. If a count specifier is used for the **s** or **a** command, then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command "./".

The **sh**(C) metacharacters * and ? may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified, then only variables local to that procedure are matched. To match only global variables, the form :*pattern* is used.

linenumber? *lm*
variable:? *lm*
: Print the value at the address from *a.out* or text space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is **i**.

variable=*lm*
linenumber=*lm*
number=*lm*
: Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then **lx** is used. The last variant of this command provides a convenient way to convert between decimal, octal, and hexadecimal.

variable!*value*
: Set *variable* to the given *value*. The value may be a number, a character constant, or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted '*character*. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

x Print the machine registers and the current machine-language instruction.

X Print the current machine-language instruction.

The commands for examining source files are:

e [*procedure*]
e [*filename*]
e [*directory/*]
e [*directory filename*]
The first two forms set the current file to the file containing *procedure* or to *filename*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory* (the default is the current working directory). The latter two forms change the value of *directory*. If no *procedure*, *filename*, or *directory* is given, the current procedure and file names are reported.

/regular expression[/]
Search forward from the current line for a line containing a string matching *regular expression* as in **ed**(C). The trailing / may be omitted.

?regular expression[?]
Search backward from the current line for a line containing a string matching *regular expression* as in **ed**(C). The trailing ? may be omitted.

p Print the current line.

z Print the current line followed by the next 9 lines. Set the current line to the last line printed.

w Window. Print the 10 lines around the current line.

number
Set the current line to the given line number. Print the new current line.

count[+-]
Increment (+) or decrement (-) the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

[*count*] **r** *args*
[*count*] **R**
Run the program with the given arguments. The **r** command with no arguments reuses the previous arguments to the program while the **R** command runs the program with no arguments. An argument beginning with < or > causes redirection for standard input or output, respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

linenumber **c** [*count*]
linenumber **C** [*count*]
Continue after a breakpoint or interrupt. If *count* is given, the program will stop when *count* breakpoints have been encountered. The signal

which caused the program to stop is reactivated with the **C** command and ignored with the **c** command. If a line number is specified, a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

linenumber **g** [*count*]
Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

s [*count*]
S [*count*]
Single-step the program through *count* lines. If no *count* is given, then the program is run for one line. **S** is equivalent to **s** except it steps through procedure calls.

i
I Single-step by one machine-language instruction. The signal which caused the program to stop is reactivated with the **I** command and ignored with the **i** command.

variable: **m** [*count*]
address: **m** [*count*]
Single-step (as with **s**) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

[*level*] **v**
Toggle verbose mode, for use when single-stepping with the **S**, **s**, or **m** commands. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

k Kill the program being debugged.

procedure(*arg1*,*arg2*,...)
procedure(*arg1*,*arg2*,...)/*m*
Execute the named procedure with the given arguments. Arguments can be integer, character, or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to **d**. This facility is only available if the program was compiled with the **-g** or **-Zi** options.

[*linenumber*] **b** [*command;command;*...]
Set a breakpoint at the given line. If a procedure name without a line number is given (for example, "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the **-g** or **-Zi** options. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the

breakpoint and control is returned to **sdb**. Otherwise the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If the **k** command is specified, control returns to **sdb** at the breakpoint.

B Print a list of the currently active breakpoints.

[*linenumber*] **d**
Delete a breakpoint at the given line. If no *linenumber* is given, then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from standard input. If the line begins with a **y** or **d**, then the breakpoint is deleted.

D Delete all breakpoints.

l Print the last executed line.

linenumber **a**
Announce. If *linenumber* is of the form *proc:number*, the command effectively does a *linenumber* **b l**. If *linenumber* is of the form *proc:*, the command effectively does a *proc*:**b T**.

Miscellaneous commands:

! *command*
The command is interpreted by **sh**(C).

new-line
If the previous command printed a source line, then advance the current line by one and print the new current line. If the previous command displayed a memory location, display the next memory location.

end-of-file character
Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last. The end-of-file character is usually ⟨Ctrl⟩-D.

< *filename*
Read commands from *filename* until the end of file is reached, and then continue to accept commands from standard input. When **sdb** is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; "<" may not appear as a command in a file.

M Print the address maps.

M [?/] [*] *b e f*
Record new values for the address map. The arguments "?" and "/" specify the text and data maps, respectively. The first segment (*b1, e1, f1*) is changed unless "*" is specified; in which case, the second segment (*b2, e2, f2*) of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

" *string*
Print the given string. The C escape sequences of the form *\character* are recognized, where *character* is non-numeric.

q Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

V Print the version number.

Q Print a list of procedures and files being debugged.

Y Toggle debug output.

Files

a.out
core

Warnings

When **sdb** prints the value of an external variable for which there is no debugging information, a warning is printed before the value. The size is assumed to be *int* (integer).

Data which are stored in text sections are indistinguishable from functions.

Line number information in *optimized* functions is unreliable, and some information may be missing.

Notes

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

See also

a.out(FP), **cc**(CP), **core**(FP), **sh**(C), **syms**(FP)

Standards conformance

sdb is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

size

print section sizes in bytes of COFF files

Syntax

size [-n] [-f] [-d] [-o] [-x] [-V] *files*

Description

The **size** command produces section size information in bytes for each loaded section in a COFF file. The size of the *.text*, *.data*, and *.bss* (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. For an archive file, information for all archive members is displayed.

The **-n** option includes NOLOAD sections in the size calculation.

The **-f** option produces full output, that is, it prints the size of every loaded section, followed by the section name in parentheses.

The **-d** option causes the sizes to be printed in decimal. This is the default behavior.

The **-o** and **-x** options cause the sizes to be printed in octal or in hexadecimal, respectively.

The **-V** flag will supply the version information on the **size** command.

Diagnostics

size: name: cannot open	if *name* cannot be read.
size: name: bad magic	if *name* is not an appropriate common object file.

Notes

If you are using XENIX binaries, see **size**(XNX) for information on appropriate usage.

Since the size of *.bss* sections is not known until link-edit time, the **size** command will not give the true total size of pre-linked objects.

This utility determines whether an object file is in OMF or COFF format and executes the appropriate binaries in each case. Since this utility relies on the files in */lib/coff* and */lib/xout*, changing or moving any of those files results in an error.

See also

a.out(FP), **ar**(FP), **as**(CP), **cc**(CP), **ld**(CP)

Standards conformance

size is conformant with:
AT&T SVID Issue 2.

strip

strip symbol and line number information from a common object file

Syntax

strip [-b] [-l] [-r] [-V] [-x] ***filename*** ...

Description

The **strip** command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using any of the following options:

-b Same as the **-x** option, but also do not strip scoping information (for example, beginning and end of block delimiters).

-l Strip line number information only; do not strip any symbol table information.

-r Do not strip static or external symbol information, or relocation information.

-V Print the version of the strip command executing on the standard error output.

-x Do not strip static or external symbol information.

If there are any relocation entries in the object file and any symbol table information is to be stripped, **strip** will complain and terminate without stripping ***filename*** unless the **-r** option is used.

If the **strip** command is executed on a common archive file (see *ar*(FP)) the archive symbol table will be removed. The archive symbol table must be restored by executing the **ar**(CP) command with the **-s** option before the archive can be link-edited by the **ld**(CP) command. **strip** will produce appropriate warning messages when this situation arises.

The **strip** command is used to reduce the file storage overhead taken by the object file.

Notes

If you are using XENIX binaries, please refer to the XENIX manual page for this utility for information on appropriate usage with XENIX binaries.

Files

TMPDIR/strp* Temporary files.

TMPDIR is usually */usr/tmp* but can be redefined by setting the environment variable **TMPDIR** (see **tempnam()** in **tmpnam**(S)).

Diagnostics

strip: name: cannot open
if *name* cannot be read.

strip: name: bad magic
if *name* is not an appropriate common object file.

strip: name: relocation entries present; cannot strip
if *name* contains relocation entries and the **-r** flag is not used, the symbol table information cannot be stripped.

Notes

This utility determines whether an object file is in OMF or COFF format and executes the appropriate binaries in each case. Since this utility relies on the files in */lib/coff* and */lib/xout*), changing or moving any of those files results in an error.

See also

a.out(FP), **ar**(CP), **ar**(FP) **as**(CP), **cc**(CP), **ld**(CP), **tmpnam**(S)

Standards conformance

strip is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

tsort

topological sort

Syntax

tsort [*file*]

Description

The **tsort** command produces on standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, standard input is understood.

The input consists of pairs of items (non-empty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

Diagnostics

Odd data: there is an odd number of fields in the input file.

See also

lorder(CP)

Standards conformance

tsort is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

unget

undo a previous get of an SCCS file

Syntax

unget [-r*SID*] [-s] [-n] *files*

Description

The **unget** command undoes the effect of a **get -e** done prior to creating the intended new delta. If a directory is named, **unget** behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of "-" is given, standard input is read with each line being taken as the name of an SCCS file to be processed.

Arguments apply independently to each named file.

-r*SID* Uniquely identifies which delta is no longer intended. (This would have been specified by **get** as the *new delta*.) The use of this option is necessary only if two or more outstanding **gets** for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

-s Suppresses printout, on standard output, of the intended delta's *SID*.

-n Causes the retention of the gotten file which would normally be removed from the current directory.

See also

delta(CP), **get**(CP), **sact**(CP)

Standards conformance

unget is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

val

validate SCCS file

Syntax

val [-] [-s] [-r*SID*] [-m*name*] [-y*type*] *files*
val -

Description

The **val** command determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to **val** may appear in any order.

The **val** command has a special argument, -, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

This permits one **val** command with different values for key letters and file arguments. For example:

```
val --yc -mabc s.abc -mxyz -ypl1 s.xyz
```

val first checks whether file *s.abc* has a value *c* for its type flag and value *abc* for the module name flag. Once this is done, **val** processes the remaining file, in this case, *s.xyz*.

val generates diagnostic messages on standard output for each command line and file processed, and also returns a single 8-bit exit code as described below.

The effects of any argument apply independently to each named file on the command line. The arguments are defined as follows:

-s The presence of this argument silences the diagnostic message normally generated on standard output for any error detected while processing each named file on the command line.

-r*SID* The argument value *SID* (SCCS identification string) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (for example, -r1 is ambiguous because it does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (for example, -r1.0 and -r1.1.0 are invalid because neither is a valid delta number). If the *SID* is valid and unambiguous, a check is made to determine if it actually exists.

-m*name* The argument value *name* is compared with the SCCS %M% keyword in *file*.

-y*type* The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by **val** is a disjunction of the possible errors, that is, it can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **-y** mismatch;
bit 7 = %M%, **-m** mismatch;

Note that **val** can process two or more files on a given command line and in turn can process multiple command lines (when reading standard input). In these cases an aggregate code is returned - a logical *OR* of the codes generated for each command line and file processed.

Diagnostics

Use **help**(CP) for explanations.

Notes

val can process up to 50 files on a single command line. Any number above 50 will produce a *core* dump.

See also

admin(CP), **delta**(CP), **get**(CP), **help**(CP), **prs**(CP)

Standards conformance

val is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

vc

version control (SCCS)

Syntax

vc [-a] [-c[*char*]] [-s] [-t] [*keyword=value* ... *keyword=value*]

Description

The **vc** command copies lines from standard input to standard output under control of its ***arguments*** and ***control statements*** encountered in standard input. In the process of performing the copy operation, user declared ***keywords*** may be replaced by their string ***value*** when they appear in plain text and/or control statements.

The copying of lines from the standard input to standard output is conditional, based on tests (in control statements) of ***keyword values*** specified in control statements or as **vc** command arguments.

A control statement is a single line beginning with a control character, except as modified by the **-t** option (see below). The default control character is a colon (":"), except as modified by the **-c** option (see below). Input lines beginning with a backslash ("\") followed by a control character are not control lines and are copied to standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A ***keyword*** is composed of nine or fewer alphanumerics; the first must be alphabetic. A ***value*** is any ASCII string that can be created with **ed**(C); a numeric ***value*** is an unsigned string of digits. ***keyword values*** may not contain blanks or tabs.

The **vc** command is **awk**-like tool used for version control of sets of files. While it is distributed as part of the SCCS package, it does not require the files it operates on to be under SCCS control.

Replacement of ***keywords*** by ***values*** is done whenever a ***keyword*** surrounded by control characters is encountered on a version control statement. The **-a** option (see below) forces replacement of ***keywords*** in *all* lines of text. An uninterpreted control character may be included in a ***value*** by preceding it with "\". If a literal "\" is desired, then it too must be preceded by "\".

Arguments

-a Forces replacement of ***keywords*** surrounded by control characters with their assigned ***values*** in *all* text lines and not just in **vc** statements.

-t All characters from the beginning of a line up to and including the first ⟨Tab⟩ character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the ⟨Tab⟩ are discarded.

-c*char* Specifies a control character to be used in place of ":".

-s Silences warning messages (not error messages) that are normally printed on the diagnostic output.

Version Control Statements

:dcl ***keywords***[, . . . , ***keywords***]
Used to declare ***keywords***. All ***keywords*** must be declared.

:asg ***keywords=values***
Used to assign ***values*** to ***keywords***. An **asg** statement overrides the assignment for the corresponding ***keyword*** on the **vc** command line and all previous **asg**'s for that ***keyword***. ***keywords*** declared, but not assigned ***values*** have null ***values***.

The following construct is used to skip lines of the standard input:

```
:if condition
:end
```

If ***condition*** is true, all lines between the **if** statement and the matching **end** statement are copied to standard output. If ***condition*** is false, all intervening lines are discarded, including control statements. Intervening **if** statements and matching **end** statements are recognized solely for the purpose of maintaining the proper **if-end** matching.

The syntax of a ***condition*** is:

```
<cond> ::= [ "not" ] <or>
<or>   ::= <and> | <and> "|" <or>
<and>  ::= <exp> | <exp> "&" <and>
<exp>  ::= "(" <or> ")" | <value> <op> <value>
<op>   ::= "=" | "!=" | "<" | ">"
<value>        ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than

< less than

( ) used for logical groupings

not may only occur immediately after the **if**, and when present, inverts the ***values*** of the entire ***condition***.

The "> " and "<" operate only on unsigned integer ***values*** (for example, : 012 > 12 is false). All other operators take strings as arguments (for example, : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > < all of equal precedence

&

|

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

::text Used for ***keyword*** replacement on lines copied to standard output. The two leading control characters are removed, and ***keywords*** surrounded by control characters in text are replaced by their ***values*** before the line is copied to the output file. This action is independent of the **-a** option.

:on
:off Turn ***keywords*** replacement on or off for all lines.

:ctl ***char*** Change the control character to ***char***.

:msg ***message***
Prints the given ***message*** on the diagnostic output.

:err ***message***
Prints the given ***message*** followed by:

ERROR: err statement on line ...

on the diagnostic output. **vc** halts execution and returns an exit code of 1.

Exit codes

0 - normal
1 - any error

See also

ed(C)

what

identify SCCS files

Syntax

what [-s] *files*

Description

The **what** command searches the given *files* for all occurrences of the pattern that **get**(CP) substitutes for %Z% (@(#)) and prints out what follows until the first occurrence of any of the following: ", >, new-line, \, or null character. For example, assuming that a C program in file *f.c* contains:

```
char ident[] = "@(#)identification information";
```

If *f.c* is compiled to yield *f.o* and *a.out*, then the command:

```
what -s f.c f.o a.out
```

will print

```
f.c:    identification information
f.o:    identification information
a.out:  identification information
```

The **what** command is intended to be used in conjunction with the SCCS command **get**(CP), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option is accepted:

-s Quit after finding the first occurrence of pattern in each file.

Diagnostics

Exit status is 0 if any matches are found, otherwise 1.

Notes

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

See also

get(CP)

Standards conformance

what is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

x286emul

emulate XENIX 80286

Syntax

x286emul [*arg* . . .] *prog286*

Description

x286emul is an emulator that allows programs from XENIX System V/286 Release 2.3 or XENIX System V/286 Release 2.3.2 on the Intel 80286 to run on the Intel 80386 processor under System V Release 3.2.

The system recognizes an attempt to **exec**(S) a 286 program, and automatically **exec**'s the 286 emulator with the 286 program name as an additional argument. It is not necessary to specify the **x286emul** emulator on the command line. The 286 programs can be invoked using the same command format as on the XENIX System V/286.

x286emul reads the 286 program's text and data into memory and maps them through the LDT (via **sysi86**(S)) as 286 text and data segments. It also fills in the jam area, which is used by XENIX programs to do system calls and signal returns. **x286emul** starts the 286 program by jumping to its entry point.

When the 286 program attempts to do a system call, **x286emul** takes control. It does any conversions needed between the 286 system call and the equivalent 386 system call, and performs the 386 system call. The results are converted to the form the 286 program expects, and the 286 program is resumed.

The following are some of the differences between a program running on a 286 and a 286 program using **x286emul** on a 386:

- Attempts to unlink or write on the 286 program will fail on the 286 with ETXTBSY. Under **x286emul**, they will not fail.
- **ptrace**(S) is not supported under **x286emul**.
- The 286 program must be readable for the emulator to read it.

Files

/bin/x286emul The emulator must be named *x286emul* and be in */bin* if it is to be automatically invoked when **exec**(S) is used on a 286 program.

xstr

extracts strings from C programs

Syntax

xstr [**-c**] [**-**] [*file*]

Description

xstr maintains a file, *strings*, into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

```
xstr -c name
```

will extract the strings from the C source in *name*, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of **xstr** is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled, a file *xs.c* declaring the common **xstr** space can be created by a command of the form

```
xstr name1 name2 name3 ...
```

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

xstr can also be used on a single file. A command

```
xstr name
```

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run **xstr** after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. **xstr** reads from its standard input when the argument "**-**" is given. An appropriate command sequence for running **xstr** after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

xstr does not touch the file *strings* unless new items are added, thus **make** can avoid remaking *xs.o* unless truly necessary.

Files

strings	Data base of strings.
x.c	Massaged C source.
xs.c	C source for definition of array **xstr**.
/tmp/xs*	Temp file when **xstr** *name* doesn't touch *strings*.

Notes

If a string is a suffix of another string in the data base, but the shorter string is seen first by **xstr**, both strings will be placed in the data base when just placing the longer one there will do.

See also

mkstr(CP)

Standards conformance

This utility was developed at the University of California at Berkeley and is used with permission.

xtil

XTI library trace control

Syntax

xtil [*-cdmv*] *file* ...

Description

The trace mechanism of the XTI library is activated and controlled via the environment variable **XTITRACE**. The trace entries of a process are collected in compressed, binary format in a dynamically created buffer and are periodically written to temporary files. These files are edited in a separate step using **xtil**.

Controlling the trace mechanism - XTITRACE

Every initial XTI call issued by a process evaluates the environment variable **XTITRACE** and, when appropriate, activates the trace mechanism. Following activation of the trace mechanism, the temporary file *XTIF<pid>* with the process ID **<pid>** is opened, if it has not already been opened. The files are granted the access permissions `rw-------` (0600). The files are located under the login name of the process. Storage is then dynamically reserved for buffering the trace entries. Storage and files are reserved for the life of the process.

The options specified in **XTITRACE** control the trace mechanism. Options **-s** and **-S** determine the extent of logging. Options **-p** and **-r** control buffering and cyclical overwriting of the file. The syntax for setting the **XTITRACE** environment variable is:

```
XTITRACE="[-sS] [-p fac] [-r wrap]"
export XTITRACE
```

The options to **XTITRACE** are:

-s The command names the values of the arguments, and the return values of the commands are logged. In the event of error, **t_errno**, **errno**, and the error position **errpos** in the library are output.

-S The same log is kept as for option **-s**. In addition, for arguments which involve pointers, the data structures addressed by the pointers are logged.

Only one of the options **-s** and **-S** may be specified. To activate the trace mechanism one of these two values **must** be specified.

-p fac The decimal digit **fac** determines the buffering factor. The buffer size is **fac** * **BUFSIZE**, with **BUFSIZE** as defined in *stdio.h*. If 0 is specified for **fac**, every trace entry is written immediately to the file (unbuffered). Any value specified for **fac** exceeding 8 is automatically reduced to 8.

If the **-p** option is not specified, **fac** defaults to 1.

-r wrap The decimal number **wrap** specifies that logging is to be directed to the second temporary file *XTIS<pid>* after **wrap** * **BUFSIZE** (with **BUFSIZE** as defined in *stdio.h*). The second file *XTIS<pid>* is handled by the trace mechanism in the same way as *XTIF<pid>*. After every **wrap** * **BUFSIZE** bytes, the trace mechanism switches between *XTIF<pid>* and *XTIS<pid>*. When this is done the old contents of the second file are lost.

If **-r** is not specified, **wrap** defaults to 512.

Editing the trace information - xtil

xtil reads the entries generated by the trace mechanism from the temporary file *file*, processes them in accordance with the specified options and outputs the results on standard output.

The options specify which trace entries from *file* are to be edited. More than one of the values may be specified per **xtil** call. If no option is specified, **-cdm** is assumed.

-c Editing is performed for the XTI calls for attaching and detaching the TS application, and for connection setup and disconnection. These are calls **t_accept()**, **t_bind()**, **t_close()**, **t_connect()**, **t_listen()**, **t_open()**, **t_rcvconnect()**, **t_rcvdis()**, **t_rcvrel()**, **t_snddis()**, **t_sndrel()**, and **t_unbind()**.

-d Editing is performed for the XTI calls for data exchange. These are calls **t_rcv()**, **t_rcvudata()**, **t_rcvuderr()**, **t_snd()**, and **t_sndudata()**.

-m Editing is performed for the other XTI calls that are not edited when options **-c** and **-d** are specified. These are calls **t_alloc()**, **t_error()**, **t_free()**, **t_getaddr()**, **t_getinfo()**, **t_getloc()**, **t_getname()**, **t_getstate()**, **t_look()**, **t_optmgmt()**, and **t_sync()**.

-v Detailed editing is performed for the XTI calls, their arguments, and the options. For arguments transferred as pointers, the addressed data structures are also output. The extent of editing of the data depends on the options specified in **XTITRACE**. If only **-v** is specified, this is the same as specifying **-cdmv**.

file ... The name of one or more files with binary trace entries to be edited.

Output format for XTI library trace function

The trace information edited by **xtil** always starts with two header lines, for example:

```
XTI    TRACE (V1.0)                    Fri Aug 12 15:13:34 1990
OPTIONS 'cmdv' , TRACE FILE 'XTIFa00963'
```

The two header lines are output once at the start of the output of the trace information. They contain:

- the version of the XTI command library,
- the date and time tracing was started,
- the editing options selected, and
- the name of the edited trace file.

Trace information for the individual XTI calls is output over several lines, the different lines having different formats. The number of lines depends on the options specified for **XTITRACE** and **xtil**.

The first line is always output. It contains the following information:

- A timestamp at the start of the first line.
- The logged XTI call (t_xxxxx).
- The arguments and their values (enclosed in parentheses), in the order required by XTI. The arguments are shown in decimal form (%d), hexadecimal form (0x%x), or symbolic form (%s). The hexadecimal form is preceded by "0x".

When interpreting the logged values, note the following:

- Arguments involving addresses are shown in the form (0x%x).
- For arguments of type integer (*.len, *.maxlen), the corresponding values are shown in form 0x%x, %d, or %s, separated by a space from the argument name.

For commands, the processing of which depends on the file mode, it is also stated whether the access was blocking (specification: BLOCK) or non-blocking (specification: NBLOCK).

The following lines are only output when option **-v** is specified for **xtil** and the trace facility has collected suitable information (option **-S** for **XTITRACE**).

In these lines, the arguments involving pointers also have the data structures they address displayed. The values of the structure components are edited in hexadecimal and in plain text. The naming of arguments and structure components is in compliance with the *X/Open Portability Guide 3*.

The following applies to marking structure components:

> The TS application must assign the component a legal value.

< The XTI functions assign the component a legal value if the command runs without error.

- The value of the component has no significance for the logged XTI call.

--- If "---" is specified instead of the value of a component, the component does not contain a legal value.

The last line of the output for an XTI call contains the **return** value. In the event of error, **t_errno** (if applicable), **errno**, and information on the error location in the library (**errpos**) are output.

Example

The example below shows the detailed log for an XTI call.

```
08:24:16 t_bind (fd  5, req 0x8054ac8, ret  0x0)
         req:        addr.maxlen(-)  addr.len(>)      addr.buf(>)
                     ---              24               0x8054d48
             0  01001800 0e000000  00000004 04003234    |                24|
            10  39370000 00000000                       |97                 |
                      qlen (>) 10
                      qlen (<) 0
         return: 0
```

Files

XTIF<pid> Files with compressed trace entries.

XTIS<pid>

Diagnostics

If the command terminates successfully, the exit status is zero, otherwise it is non-zero.

yacc

yet another compiler-compiler—a parser generator

Syntax

yacc [-vdlt] [-S[ailmnrsw[*num*]]] ***filename***

Description

The **yacc** command is used to generate C code that implements a parser. The program that **yacc** generates is based on an input file consisting of a grammar and possibly some C code fragments and internal declarations. **yacc** converts a context-free grammar into a set of tables for a simple automaton which executes a parsing algorithm. Code that is executed whenever a grammar production is recognized may be supplied. The grammar may be ambiguous; precedence rules are used to resolve ambiguities.

The output file containing the parser code is called *y.tab.c*. This program must be compiled by the C compiler; the parsing routine is called **yyparse**. This program must be linked with a lexical analyzer routine with the name **yylex**, as well as **main** and **yyerror**, an error-handling routine. **lex**(CP) will generate a lexical analyzer routine called **yylex**; if the program is linked with the **yacc** library, **main** and **yyerror** will be supplied.

Options

-d A file called *y.tab.h* will be generated. This file contains **#define** statements associating **yacc**-assigned token codes with the user-declared token names. This file may be included in other programs which need to access the token codes.

-l The code produced in *y.tab.c* will not contain any **#line** constructs. This option should only be used after the grammar and the associated actions are fully debugged.

-S This flag is used to increase the allocation of various resources. It allows you to set the following parameters:

Key letter	Name	Meaning	Default
a	ACTSIZE	action table storage size	10,000
m	MEMSIZE	production storage size	20,000
s	NSTATES	state table size	1,250
r	NPROD	number of grammar rules	600
n	NONTERM	number of non-terminals	600
i	CNAMSZ	number of idents and literals	4,000
l	LSETSIZE	number of look-ahead sets	700
w	WSETSIZE	number of working sets	250

For example:

```
yacc -Sw600 yacc.y
```

The number following the key letter is an integer increasing the allocation of the corresponding resource. **-S** can appear several times on the command line.

-t Runtime debugging code is always generated in *y.tab.c*. By default, this code is not compiled. However, when **yacc**'s **-t** option is used, this debugging code will be compiled. Independent of whether the **-t** option is used, the runtime debugging code is under the control of *YYDEBUG*, a preprocessor symbol. If *YYDEBUG* has a non-zero value, then the debugging code is compiled. If its value is zero, then the code will not be compiled. A program compiled without the runtime debugging code will be smaller and slightly faster.

-v A file called *y.output* will be generated. This file contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

Files

y.output	
y.tab.c	
y.tab.h	Defines for token names.
yacc.tmp, yacc.debug, yacc.acts	Temporary files.
/usr/lib/yaccpar	Parser prototype for C programs.

Diagnostics

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the *y.output* file. If some rules are not reachable from the start symbol, this is reported.

Caveat

Because file names are fixed, only one **yacc** process can be active in a given directory.

See also

lex(CP)

Standards conformance

yacc is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

Files for Programming (FP)

Intro

introduction to file formats for programmers

Description

This section outlines the formats of various files. The C structure declarations for the file formats are given where applicable. Usually, these structures can be found in the directories */usr/include* or */usr/include/sys*

The following is a listing of the manual pages found in this section and a brief description of the information in each.

Manual page	Description
86rel(FP)	Intel 8086 Relocatable Format for Object Modules
acct(FP)	format of per-process accounting file
a.out(FP)	UNIX common assembler and link editor output
ar(FP)	archive file format
core(FP)	format of core image file
dirent(FP)	filesystem-independent directory entry
dir(FP)	format of a directory
filehdr(FP)	file header for common object files
filesystem(FP)	format of filesystem types
inode(FP)	format of an inode
Intro(FP)	introduction to file formats for programmers
langinfo(FP)	language information constants
lconv(FP)	Internationalization (native language support) information
ldfcn(FP)	common object file access routines
limits(FP)	header file for implementation-specific constants
linenum(FP)	line number entries in a common object file
mallinfo(FP)	malloc (libmalloc) information
mem(FP)	discusses access to the computer's physical and virtual memory
mfsys(FP)	configuration file for filesystem types
msg(FP)	IPC message structures
netbuf(FP)	TLI/XTI network buffer structure
nl_types(FP)	data types for native language support
nssend(FP)	name server request structure
null(FP)	the null file
passwd(FP)	password entry structure
plot(FP)	graphics interface
pnch(FP)	file format for card images
proc(FP)	process table structure
reloc(FP)	relocation information for a common object file
sccsfile(FP)	format of an SCCS file
scnhdr(FP)	section header for a common object file
scr_dump(FP)	format of curses screen image file
sem(FP)	controls structures for UNIX System V semaphores
sfsys(FP)	local filesystem type file
shm(FP)	IPC shared memory structures
stat(FP)	data returned by stat system call
syms(FP)	common object file symbol table format
t_info(FP)	TLI and XTI transport protocol structure
types(FP)	primitive system data types
unistd(FP)	file header for symbolic constants
x.out(FP)	format of XENIX link editor output

86rel

Intel 8086 Relocatable Format for Object Modules

Syntax

```
#include <sys/relsym86.h>
```

Description

Intel 8086 Relocatable Format, or **86rel**, is the object module format generated by **masm**(CP), and the input format for the linker **ld**(CP). The include file *relsym86.h* specifies appropriate definitions to access **86rel** format files from C. For the technical details of the **86rel** format, see *Intel 8086 Object Module Format External Product Specification*.

An **86rel** consists of one or more variable length records. Each record has at least three fields: the record type, length, and checksum. The first byte always denotes the record type. There are thirty-one different record types. Only eleven are used by **ld**(CP) and **masm**(CP). The word after the first byte is the length of the record in bytes, exclusive of the first three bytes. Following the length word are typically one or more fields. Each record type has a specific sequence of fields, some of which may be optional or of varying length. The last byte in each record is a checksum. The checksum byte contains the sum modulo 256 of all other bytes in the record. The sum modulo 256 of all bytes in a record, including the checksum byte, should equal zero.

With few exceptions, **86rel** strings are length prefixed and have no trailing null. The first byte contains a number between 0 and 40, which is the remaining length of the string in bytes. Although the Intel specification limits the character set to upper case letters, digits, and the characters "?", "@", ":", ".", and "_", **masm**(CP) uses the complete ASCII character set.

The Intel Object Module Format (OMF) specification uses the term "index" to mean a positive integer in either the range 0 to 127 or 128 to 32,768. This terminology is retained in this document and elsewhere in the **86rel** literature. An index has one or two bytes. If the first byte has a leading 0 bit, the index is assumed to have only one byte, and the remainder of the byte represents a positive integer between 0 and 127. If the second byte has a leading 1 bit, the index is assumed to take up two bytes, and the remainder of the word represents a positive integer between 128 and 32,768.

Following is a list of record types and the hexadecimal values of their first byte, as defined in *relsym86.h*.

```
#define MRHEADR 0x6e    /* rel module header */
#define MREGINT 0x70    /* register initialization */
#define MREDATA 0x72    /* explicit (enumerated) data image */
#define MRIDATA 0x74    /* repeated (iterated) data image */
#define MOVLDEF 0x76    /* overlay definition */
#define MENDREC 0x78    /* block or overlay end record */
#define MBLKDEF 0x7a    /* block definition */
#define MBLKEND 0x7c    /* block end */
#define MDEBSYM 0x7e    /* debug symbols */
#define MTHEADR 0x80    /* module header, usually first in a rel file */
#define MLHEADR 0x82    /* link module header */
#define MPEDATA 0x84    /* absolute data image */
#define MPIDATA 0x86    /* absolute repeated (iterated) data image */
#define MCOMENT 0x88    /* comment record */
#define MMODEND 0x8a    /* module end record */
#define M386END 0x8b    /* 32 bit module end record */
#define MEXTDEF 0x8c    /* external definition */
#define MTYPDEF 0x8e    /* type definition */
#define MPUBDEF 0x90    /* public definition */
#define MPUB386 0x91    /* 32 bit public definition */
#define MLOCSYM 0x92    /* local symbols */
#define MLOC386 0x93    /* 32 bit local symbols */
#define MLINNUM 0x94    /* source line number */
#define MLIN386 0x95    /* 32 bit source line number */
#define MLNAMES 0x96    /* name list record */
#define MSEGDEF 0x98    /* segment definition */
#define MSEG386 0x99    /* 32 bit segment definition */
#define MGRPDEF 0x9a    /* group definition */
#define MFIXUPP 0x9c    /* fix up previous data image */
#define MFIX386 0x9d    /* fix up previous 32 bit data image */
#define MNONE1  0x9e    /* none */
#define MLEDATA 0xa0    /* logical data image */
#define MLED386 0xa1    /* 32 bit logical data image */
#define MLIDATA 0xa2    /* logical repeated (iterated) data image */
#define MLID386 0xa3    /* 32 bit logical repeated (iterated) data image */
#define MLIBHED 0xa4    /* library header */
#define MLIBNAM 0xa6    /* library names record */
#define MLIBLOC 0xa8    /* library module locations */
#define MLIBDIC 0xaa    /* library dictionary */
#define MCOMDEF 0xb0    /* communal names definition */
#define MBAKPAT 0xb2    /* backpatch record */
#define MBAK386 0xb3    /* 32 bit backpatch record */
#define MLOCEXD 0xb4    /* external definition visible within module only */
#define MLOCPUB 0xb6    /* public definition visible within module only */
#define MLPB386 0xb7    /* 32 bit public visible within module only */
#define MLOCCOM 0xb8    /* communal name visible within module only */
#define MLIBHDR 0xf0    /* library header */
#define MLIBDHD 0xf1    /* library dictionary header */
```

In the following discussion, the main features of each record type are given. If the record is not used by either **masm**(CP) or **ld**(CP), it is not listed.

THEADR — The record type byte is 0x80. The THEADR record specifies the name of the source module at assembly time (see Notes). The only field is the T-MODULENAME, which contains a length-prefixed string derived from the base name of the source module.

COMENT — The record type byte is 0x88. The COMENT record may contain a remark generated by the compiler system. **masm**(CP) inserts the string "UNIX 8086 ASSEMBLER."

MODEND — The record type byte is 0x8a. The MODEND record terminates a module. It can specify whether the current module is to be used as the entry point to the linked executable. If the module is an entry point, the MODEND record can then specify the address of the entry point within the executable.

EXTDEF — The record type byte is 0x8c. The EXTDEF record contains the names and types of symbols defined in other modules by a PUBDEF record (see below). This corresponds to the C storage class "extern." The fields consist of one or more length-prefixed strings, each followed by a type index. The indices reference a TYPDEF record seen earlier in the module. **masm**(CP) generates only one EXTDEF per exterior symbol.

TYPDEF — The record type byte is 0x8e. The TYPDEF record gives a description of the type (size and storage attributes) of an object or objects. This description can then be referenced by EXTDEF, PUBDEF, and other records.

PUBDEF — The record type byte is 0x90. The PUBDEF record gives a list of one or more names that may be referenced by other modules at link time ("publics"). The list of names is preceded by group and segment indices, which reference the location of the start of the list of publics within the current segment and group. If the segment and group indices are zero, a frame number is given to provide an absolute address in the module. The list consists of one or more of length-prefixed strings, each associated with a 16-bit offset within the current segment and a type index referring to a TYPDEF.

LNAMES — The record type byte is 0x96. The LNAMES record gives a series of length-prefixed strings which are associated with name indices within the current module. Each name is indexed in the sequence given, starting with 1. The names may then be referenced within the current module by successive SEGDEF and GRPDEF records to provide strings for segments, classes, overlays or groups.

SEGDEF — The record type byte is 0x98. The SEGDEF record provides an index for referencing a segment, and information concerning segment addressing and attributes. This index may be used by

other records to refer to the segment. The first byte in the record after the length field gives information about the alignment and about combination attributes of the segment. The next word is the segment length in bytes. Segments are restricted to a maximum of 65,536 bytes in length. Following this word is an index for the segment. In addition, the SEGDEF can optionally end with class and/or overlay index fields.

GRPDEF — The record type is 0x9a. The GRPDEF record provides a name to reference several segments. The group name is implemented as an index.

FIXUPP — The record byte is 0x9c. The FIXUPP record specifies one or more load-time address modifications ("fixups"). Each fixup refers to a location in a preceding LEDATA (see below) record. The fixup is specified by four data: a location, a mode, a target and a frame. The frame and target may be specified explicitly or by reference to an already defined fixup.

LEDATA — The record type byte is 0xa0. This record provides a contiguous text or data image which the loader **ld**(CP) uses to construct a portion of an 8086 run-time executable. The image might require additional processing (see FIXUPP) before being loaded into the executable. The image is preceded by two fields, a segment index and an enumerated data offset. The segment index (see INDEX) specifies a segment given by a previously seen SEGDEF. The enumerated data offset (a word) specifies the offset from the start of this segment.

Notes

If you attempt to load a number of modules assembled under the same basename, the loader will try to put them all in one big segment. In 286 programs, segment size is limited to 64K. In a large program, the resulting segment size can easily exceed 64K. A large model code executable results from the link of one or more modules, composed of segments that aggregate into greater than 64K of text.

Be sure that the assembly-time name of the module has the same basename as the source. This can occur if the source module is preprocessed not by **cc**(CP), but, for example, by hand or shell script, prior to assembly. The following example is incorrect:

```
#incorrect
cc -E module1.c | filter > x.c
cc x.c
mv x.o module1.o
cc -E module2.c | filter > x.c
cc x.c
mv x.o module2.o
cc -E module3.c | filter > x.c
cc x.c
mv x.o module3.o
ld module1.o module2.o module3.o
```

Each module should have a unique name when assembled, as follows:

```
#correct
cc -E module1.c | filter > x.c
cc -S x.c
mv x.s module1.s
as module1.s
ld module1.o module2.o module3.o
```

See also

as(CP), **ld**(CP)

Standards conformance

86rel is an extension of AT&T System V provided by Santa Cruz Operation.

a.out

UNIX common assembler and link editor output

Syntax

```
#include <a.out.h>
```

Description

The file name *a.out* is the default output file name from the link editor **ld**(CP). The link editor will make *a.out* executable if there were no errors in linking. The output file of the UNIX assembler **as**(CP) also follows the common object file format of the *a.out* file, although the default file name is different.

A common object file consists of a file header, a UNIX system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is as shown:

File header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
String table.

The last three parts of an object file (line numbers, symbol table and string table) may be missing if the program was linked with the **-s** option of **ld**(CP) or if they were removed by **strip**(CP). Also note that the relocation information will be absent after linking unless the **-r** option of **ld**(CP) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, as discussed later) are in bytes.

When an *a.out* file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, with the latter actually being initialized to all 0's), and a stack. The text segment starts at location virtual address 0.

The *a.out* file produced by **ld**(CP) may have one of two magic numbers in the first field of the UNIX system header. A magic number of 0410 indicates that the executable must be swapped through the private swapping store of the UNIX system, while the magic number 0413 causes the system to attempt to page the text directly from the a.out file.

In a 0410 executable, the text section is loaded at virtual location 0x00000000. The data section is loaded after the text section.

For a 0413 executable, the headers (file header, UNIX system header, and section headers) are loaded at the beginning of the text segment, and the text follows the headers in the user address space. The first text address will equal the sum of the sizes of the headers, and will vary, depending on the number of sections in the *a.out* file. In an *a.out* file with three sections (*.text*, *.data*, and *.bss*), the first text address is at 0x000000D0. The data section starts in the next page table directory after the last one used by the text section, in the first page of that directory, with an offset into that page equal to the first unused memory offset in the last page of text. Thus, given that *etext* is the address of the last byte of the text section, the 1st byte of the data section will be at 0x00400000 + (*etext* & 0xFFC00000) + ((*etext*+1) & 0xFFC00FFF).

On the 80386 computer the stack begins at location 7FFFFFFC and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the **brk**(S) system call.

For relocatable files, the value of a word in the text or data portions that is not a reference to an undefined external symbol is the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word; the storage class of the symbol-table entry for the symbol will be marked as an "external symbol", and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

File header

The format of the *filehdr* header is:

```
struct filehdr
{
        unsigned short      f_magic;        /* magic number */
        unsigned short      f_nscns;        /* number of sections */
        long                f_timdat;       /* time and date stamp */
        long                f_symptr;       /* file ptr to symtab */
        long                f_nsyms;        /* # symtab entries */
        unsigned short      f_opthdr;       /* sizeof(opt hdr) */
        unsigned short      f_flags;        /* flags */
};
```

UNIX system header

The format of the UNIX system header is:

```
typedef struct aouthdr
{
        short       magic;        /* magic number */
        short       vstamp;       /* version stamp */
        long        tsize;        /* text size in bytes, padded */
        long        dsize;        /* initialized data (.data) */
        long        bsize;        /* uninitialized data (.bss) */
        long        entry;        /* entry point */
        long        text_start;   /* base of text used for this file */
        long        data_start;   /* base of data used for this file */
} AOUTHDR;
```

Section header

The format of the section header is:

```
struct scnhdr
{
        char                s_name[SYMNMLEN]; /* section name */
        long                s_paddr;          /* physical address */
        long                s_vaddr;          /* virtual address */
        long                s_size;           /* section size */
        long                s_scnptr;         /* file ptr to raw data */
        long                s_relptr;         /* file ptr to relocation */
        long                s_lnnoptr;        /* file ptr to line numbers */
        unsigned short      s_nreloc;         /* # reloc entries */
        unsigned short      s_nlnno;          /* # line number entries */
        long                s_flags;          /* flags */
};
```

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format:

```
struct reloc
{
        long        r_vaddr;        /* (virtual) address of reference */
        long        r_symndx;       /* index into symbol table */
        ushort      r_type;         /* relocation type */
};
```

The start of the relocation information is `s_relptr` from the section header. If there is no relocation information, `s_relptr` is 0.

Symbol table

The format of each symbol in the symbol table is:

```
#define  SYMNMLEN       8
#define  FILNMLEN       14
#define  DIMNUM         4

struct syment
{
    union                                   /* all ways to get a symbol name */
    {
        char        _n_name[SYMNMLEN];      /* name of symbol */
        struct
        {
            long        _n_zeroes;          /* == 0L if in string table */
            long        _n_offset;          /* location in string table */
        } _n_n;
        char        *_n_nptr[2];            /* allows overlaying */
    } _n;
    long        n_value;                    /* value of symbol */
    short       n_scnum;                    /* section number */
    unsigned short      n_type;             /* type and derived type */
    char        n_sclass;                   /* storage class */
    char        n_numaux;                   /* number of aux entries */
};

#define  n_name         _n._n_name
#define  n_zeroes       _n._n_n._n_zeroes
#define  n_offset       _n._n_n._n_offset
#define  n_nptr         _n._n_nptr[1]
```

(FP)

Some symbols require more information than a single entry; they are followed by auxiliary entries that are the same size as a symbol entry. The format follows:

```
union auxent {
        struct {
                long        x_tagndx;
                union {
                        struct {
                                unsigned short        x_lnno;
                                unsigned short        x_size;
                        } x_lnsz;
                        long        x_fsize;
                } x_misc;
                union {
                        struct {
                                long        x_lnnoptr;
                                long        x_endndx;
                        }  x_fcn;
                        struct {
                                unsigned short        x_dimen[DIMNUM];
                        } x_ary;
                } x_fcnary;
                unsigned short  x_tvndx;
        } x_sym;

        union {
                char    x_fname[FILNMLEN];
                struct {
                        long x_zero;
                        long x_offset;
                } x_longname;
        } x_file;

        struct {
                long            x_scnlen;
                unsigned short  x_nreloc;
                unsigned short  x_nlinno;
        } x_scn;

        struct {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        } x_tv;
};
```

Indexes of symbol table entries begin at zero. The start of the symbol table is **f_symptr** (from the file header) bytes from the beginning of the file. If the symbol table is stripped, **f_symptr** is 0. The string table (if one exists) begins at **f_symptr** + (**f_nsyms** * SYMESZ) bytes from the beginning of the file.

See also

as(CP), **cc**(CP), **ld**(CP), **brk**(S), **filehdr**(FP), **ldfcn**(FP), **linenum**(FP), **reloc**(FP), **scnhdr**(FP), **syms**(FP)

acct

format of per-process accounting file

Description

Files produced as a result of calling **acct**(S) have records in the form defined by *<sys/acct.h>*.

In `ac_flag`, the **AFORK** flag is turned on by each **fork**(S) and turned off by an **exec**(S.) The `ac_comm` member is inherited from the parent process and is reset by any call to **exec**(S). Each time the system charges the process with a clock tick, it also adds the current process size to the `ac_mem` member, computed as follows:

(data size) + (text size) / (number of in-core processes using text)

The value of `ac_mem/ac_stime` can be viewed as an approximation to the mean process size, as modified by text-sharing.

Notes

The `ac_mem` value for a short-lived command gives little information about the actual size of the command, because `ac_mem` may be incremented while a different command (for example, the shell) is being executed by the process.

See also

acctcom(ADM), **acct**(S)

Standards conformance

acct is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

ar

archive file format

Description

The archive command **ar** is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor **ld**(CP).

A file produced by **ar** has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number string is in a portable archive format that uses the ASCII string:

```
!<arch>\n
```

The header of each file is declared in */usr/include/ar.h*.

Each file begins on a word boundary; a null byte is inserted between files if necessary. The size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

Notes

The **ar**(CP) utility now handles filenames longer than 14 characters so that the upper limit for the filename component is 512 characters and the limit for a full pathname is 1024 characters.

Files with long filenames are stored in the archive using a short filename convention: **_LNGFIL***xxxxxx*. Where the *xxxxxx* is an ASCII-encoded decimal serial number, for example, *_LNGFIL000001* and *_LNGFIL000002*. These shortened filenames are used to identify the long filenames by taking the serial numbers and using them as indexes into a map table of long filenames. Because of the encoding, up to 1,000,000 long filenames can be stored in an archive.

A table containing the mapping information on long filenames (called the **maptable**) is stored in the archive as a regular (archived) file. This file (called the ***mapfile***) is either the first or the second file in the archive depending on whether the archive file has a symbol table file or not. This ***mapfile*** is part of the archive if (and only if) there are files with long filenames.

If there are any COFF files in the archive, the first archive member is the symbols table. If there are any files with long filenames, the ***mapfile*** is the next file after the symbol table.

The format of the long filenames table is:

<string_number><serial_length><string>

Where:

- ***<string_number>*** is a 7-byte numeric ASCII string
- ***<serial_length>*** is a 7-byte numeric ASCII string
- ***<string>*** is a string of ***string_length*** characters long.

An example entry is:

```
000001900000001longfilenamemapfile
```

The string length is 19 characters, the serial number entry is 1, and the string is longfilenamemapfile.

See also

ar(CP), **ar**(S), **ld**(CP)

core

Format of core image file

Description

The Operating System writes out a core image of a terminated process when various errors occur. See **sigaction**(S) for a list of reasons; the most common errors are memory violations, illegal instructions, and user-generated quit signals. The core image is called **core** and is written in the process' working directory (provided it can be; normal access controls apply).

A process with an effective user (group) ID different from the real user (group) ID will not produce a core image. If a file named **core** already exists in the process' working directory and is not a regular file or has links, no core image is produced.

The core image contains all mutable portions of the process' virtual address space plus information from the system about the state of the process. If the text segment is read-only and shared, or separated from data space, it is not dumped. Attached shared memory, shared data, and shared library data are dumped.

The **core** file is logically divided into a series of sections. A section either describes the **core** file, contains a copy of the contents of a kernel data structure, or contains a copy of the contents of the process' memory at the time of the dump.

Each section consists of a header describing the section, optional additional information about the section's contents, and the section's contents. Consecutive sections may not be physically adjacent. Whilst any optional information physically follows the header, the contents may not. The header contains the size and offset of its contents. The headers themselves form a chain.

The section header is defined in *<sys/paccess.h>*:

```
struct coresecthead {
   uchar_t     cs_stype;  /* Type of section (CORES_xxx) */
   uchar_t     cs_hsize;  /* Size (bytes) of section header */
   ushort      cs_osize;  /* Size (bytes) of optional info */
   union {
      struct {
         short csxp_rflg; /* Associated pregion's p_flags */
         short csxp_rtyp; /* Associated pregion's p_type */
      } csx_preg;
      ulong_t  csx_magic; /* COREMAGIC_xxx for MAGIC, OFFSETS */
   } cs_x;
   ulong_t     cs_vaddr;  /* Virtual address of contents */
   ulong_t     cs_vsize;  /* Size (bytes) of contents */
   long        cs_sseek;  /* Offset of this section's contents */
   long        cs_hseek;  /* Offset of next section's header */
};
```

Each **coresecthead** structure in a **core** file contains:

cs_stype
: The type of section dumped (see list below).

cs_hsize
: The size, in bytes, of this core section header (**coresecthead** structure) including this size field itself.

cs_x
: The format and contents of this union depend on the section's type as described below.

cs_osize
: The size, in bytes, of the optional information about this core section's contents. The format of this optional additional information depends on the type of information contained in the section. The optional information starts immediately after this **coresecthead** structure in the **core** file.

cs_vaddr
: The virtual address of this section's contents in the appropriate address space.

cs_vsize
: The size, in bytes, of this section's contents.

cs_sseek
: The offset (from zero) in the **core** file of this section's contents. A section's contents may not be physically adjacent to its header and additional information.

cs_hseek
: The offset (from zero) in the **core** file of the "next" section. A zero or negative offset ends the chain. The logical order of sections in a **core** file is unrelated to the numerical values of the section header offsets.

Should a future release or update of the system require an expanded `coresecthead` structure, the additional fields will be added at the end of the structure. Programs should therefore not assume that `sizeof(struct coresecthead)` bytes were written to the core file — the `cs_hsize` field contains the number of bytes that were written.

The defined `cs_stype` section types include:

CORES_MAGIC
If this section is dumped, it is always at offset zero. This section is always dumped unless the `CORES_UAREA` section (described below) could not be dumped. The `cs_x.csx_magic` field is either `COREMAGIC_NUMBER` (this core file is believed valid) or `COREMAGIC_INVALID` (the core file may be corrupt). This section always heads the chain, and the "next" section in the chain for valid core files is always the `CORES_OFFSETS` section (described below). The additional information is a byte containing the last error encountered when writing the core file (as defined in *<sys/errno.h>*), and the contents of this section are always the null terminated name of the core file (which is always `" core "` in current releases of the Operating System).

CORES_UAREA
The system's `user` structure (defined in *<sys/user.h>*) for the process. This section is always present in a valid core file. No additional information is currently defined for this section. The `cs_x` union is reserved for possible future use.

CORES_PREGION
Memory attached to the process' virtual address space. The `cs_x.csx_preg.csxp_rtyp` and `cs_x.csx_preg.csxp_rflg` fields contain the `pregion` structure's `p_type` and `p_flags` fields (respectively) for the memory attached to the process at virtual address `cs_vaddr`. Those fields are defined in *<sys/region.h>*. The type of memory is given by `cs_x.csx_preg.csxp_rtyp` and includes `PT_STACK` (the process' stack), `PT_DATA` (the process' data), `PT_SHMEM` (an attached shared memory segment), and `PT_LIBDAT` (data associated with a shared library). Text (`PT_TEXT` and `PT_LIBTXT`) are never dumped. Only `PT_SHMEM` memory has additional information — the `shmds_id` structure associated with this shared memory (defined in *<sys/shm.h>*). No other additional information is currently defined for this section.

CORES_PROC
The system's `proc` structure as defined in *<sys/proc.h>* for this process. There is at most one such section in a valid core file. No additional information is currently defined for this section. The `cs_x` union is reserved for possible future use.

CORES_ITIMER
The process' interval timer values as an array of `itimerval` structures (defined in *<sys/itimer.h>*). There is at most one such section in a valid core file. No additional information is currently defined for this section. The `cs_x` union is reserved for possible future use.

CORES_SCOUTSNAME
The system's `scoutsname` structure defined in *<sys/utsname.h>*. There is at most one such section in a valid core file. No additional information is currently defined for this section. The `cs_x` union is reserved for possible future use.

CORES_OFFSETS
Always the physically last section in a valid core file, this section contains the process' `coreoffsets` structure defined in *<sys/paccess.h>*. The additional information's size is always an exact number of longwords. The last longword is always the core file offset (from zero) of this section's header, and the next-to-last longword is the number of sections in the chain (excluding the `CORES_MAGIC` section but including this `CORES_OFFSET` section itself). Only the `coresecthead` section may preceed this section in the chain, so this section may be considered the head of the chain. No additional longwords are currently defined. Immediately following these longwords is the `coreoffsets` structure (also defined in *<sys/paccess.h>*). That structure is always at the very end of a valid core file. The `cs_x.csx_magic` field is either `COREMAGIC_NUMBER` (the core file is believed valid) or `COREMAGIC_INVALID` (the core file may be corrupt).

The last longword of the `coreoffsets` structure is always the size of that structure itself (including that last longword):

```
struct coreoffsets {
   uoff u_info;       /* interface version (C_VERSION) */
   uoff u_user;       /* offset of upage in core file */
   uoff u_ldt;        /* offset of ldt in core file  */
   uoff u_data;       /* offset of data in core file */
   uoff u_stack;      /* offset of stack in core file */
   uoff u_usize;      /* size of uarea */
   uoff u_uaddr;      /* kernel virtual address of uarea  */
   uoff u_ar0;        /* user register save area pointer */
   uoff u_fps;        /* floating point save area  */
   uoff u_fper;       /* 2.3 field */
   uoff u_fpemul;     /* separate emulator save area  */
   uoff u_fpvalid;    /* valid save fpstate */
   uoff u_weitek;     /* flag indicating weitek usage */
   uoff u_weitek_reg; /* weitek save area */
   uoff u_ssize;      /* stack size */
   uoff u_dsize;      /* data size  */
   uoff u_tsize;      /* text size  */
   uoff u_comm;       /* name of executable */
   uoff u_ldtlimit;   /* size of ldt   */
   uoff u_sub;        /* stack upper  bound */
   uoff u_sdata;      /* 3.2 u_exdata.ux_datorg  */
   uoff u_mag;        /* u_exdata.ux_mag  */
   uoff u_sigreturn;
   uoff u_signal;
   uoff uvstack;      /* virtual address of stack top */
   uoff uvtext;       /* virtual address of text */
   uoff size;         /* size of this structure */
};
```

Some of these fields contain information about the core file's layout:

u_info
Format version number. The current version is `C_VERSION`, which is 2. This manual page describes the version 2 format.

u_user
Offset (from zero) in the core file of the process' `user` structure, which is the contents of the `CORES_UAREA` section. This field is identical to that section's `cs_sseek`

u_ldt
Offset from the beginning of the `CORES_UAREA` section's contents (that is, from the beginning of the dumped `user` structure) of the process' LDT. The number of entries in the table varies and is given by the `u_ldtlimit` field in the `user` structure, whose offset is given by the `u_ldtlimit` field of the `coreoffsets` structure.

u_data
Offset (from zero) in the core file of the `PT_DATA` `CORES_PREGION` section containing the data and BSS loaded from the process' executable file. This field is identical to that section's `cs_sseek`.

u_stack
Offset (from zero) in the core file of the process' stack, which is the contents of the `PT_STACK` `CORES_PREGION` section. This field is identical to that section's `cs_sseek`.

u_usize
The size, in bytes, of the process' `user` -area. This field is identical to the `CORES_UAREA` section's `cs_vsize`.

u_uaddr
The kernel's virtual address of the process' `user` structure. This field is identical to the `CORES_UAREA` section's `cs_vaddr`.

u_fper
Obsolete; always zero.

uvstack
The numerically highest virtual address of the process' stack. For historical reasons this does not include the last longword of the stack.

uvtext
The numerically lowest virtual address of the process' text.

The other fields in the `coreoffsets` structure are the offset (in bytes) from the beginning of the `USERS_UAREA` contents of the similarly-named fields in the dumped `user` structure:

coreoffsets field	Structure user offset	Comments
u_ar0	u_ar0	Kernel virtual address of process' registers
u_fps	u_fps	Saved FPU hardware state
u_fpemul	u_fps	Saved FPU emulator state
u_fpvalid	u_fpvalid	FPU state saved?
u_weitek	u_weitek	
u_weitek_reg	u_weitek_reg	
u_ssize	u_ssize	Pages of process' stack in **core**
u_dsize	u_dsize	Pages of process' data in **core**
u_tsize	u_tsize	Pages of process' separate/shared text
u_comm	u_comm	Executable file's basename
u_ldtlimit	u_ldtlimit	
u_sub	u_sub	Lowest address in process' stack
u_sdata	u_exdata.ux_datorg	Process' data virtual address
u_mag	u_exdata.ux_mag	Executable's magic number
u_sigreturn	u_sigreturn	
u_signal	u_signal[0]	

The size of the **core** file never exceeds the process' **ulimit**(S) in effect at the time of the core dump. If the **ulimit** is insufficient to contain the entire `user` and `coreoffsets` structures plus their corresponding `coresecthead` headers, nothing is written to the **core** file. If the **ulimit** is too small to contain both the process' stack (`PT_STACK`) and data (`PT_DATA`) memory, only the stack is written (plus the `user` and `coreoffsets` structures); if the **ulimit** is too small to contain the stack, then only the `user` and `coreoffsets` sections are written. Other sections are squeezed in as the **ulimit** permits.

Notes

Version 1 **core** files contain only the first two pages of the `user` structure (always at offset zero), the process' stack and data memory, and the `coreoffsets` structure (always at the end of the **core** file). No other memory or information are present, and no `coresecthead` structures are dumped. In particular, there is no `CORES_MAGIC` section in a genuine version 1 **core** file.

The `u_ldt` field of the `coreoffsets` structure in a genuine version 1 **core** file may point to memory which was not actually dumped, or which was only partially dumped.

The version 2 **core** file format is designed so most programs expecting a version 1 **core** file should continue to work unchanged, although they will not be able to access the additional information present in version 2 files. However, some utilities require version 1 **core** files. The **corex**(C) command converts a version 2 **core** file into its version 1 equivalent. The LDT in the converted **core** file may — like genuine version 1 **core** files — either not be dumped or be

only partially dumped.

The `CORES_MAGIC` section actually overlays the beginning of the `CORES_UAREA` section's contents so that, like version 1 **core** files, the `user` structure in version 2 **core** files starts at offset zero. There is no data of consequence in the overlaid bytes, being the furthermost extent of the kernel's stack.

The `u_comm` string in the `user` structure is not necessarily null-terminated.

The `u_ar0` field in the `coreoffsets` structure is the offset in the `CORES_UAREA` section of the kernel's virtual address which in turn points to the process' registers. That offset is not the offset of the saved registers themselves. The dumped `u_ar0` kernel virtual address refers to core dumped someplace in the `CORES_UAREA` section; the pointed to dumped core is arranged as described in *<sys/regs.h>*.

See also

Intro(S), **adb**(CP), **codeview**(CP), **corex**(C), **dbxtra**(CP), **dbXtra**(CP), **sdb**(CP), **sdget**(S), **setuid**(S), **shmctl**(S), **sigaction**(S), **signal**(S), **termio**(M), **ulimit**(S)

dir

format of a directory

Syntax

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

Description

A directory behaves like an ordinary file, except that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry (see *filesystem*(FP)). The structure of a directory is given in the include file */usr/include/sys/dir.h*.

By convention, the first two entries in each directory are "dot" (.) and "dotdot" (..). The first is an entry for the directory itself. The second is for the parent directory. The meaning of dotdot is modified for the root directory of the master file system; there is no parent, so dotdot has the same meaning as dot.

See also

filesystem(FP)

dirent

filesystem-independent directory entry

Syntax

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

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

Description

Different file system types may have different directory entries. The `dirent` structure defines a file system-independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the **getdents**(S) system call.

The `dirent` structure is defined below.

```
struct    dirent {
                ino_t          d_ino;
                short          d_pad;
                off_t          d_off;
                unsigned short d_reclen;
                char           d_name[1];
                };
```

The `d_ino` is a number which is unique for each file in the file system. The field `d_off` is the offset of that directory entry in the actual file system directory. The field `d_name` is the beginning of the character array giving the name of the directory entry. This name is null terminated and may have at most **MAXNAMLEN** characters. This results in filesystem-independent directory entries that are variable length entities. The value of `d_reclen` is the record length of this entry. This length is defined to be the number of bytes between the current entry and the next one, so that it will always result in the next entry being on a long boundary.

Files

/usr/include/sys/dirent.h

See also

getdents(S)

Standards conformance

dirent is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

filehdr

file header for common object files

Syntax

```
#include <filehdr.h>
```

Description

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct   filehdr
{
unsigned short      f_magic ;        /* magic number */
unsigned short      f_nscns ;        /* number of sections */
long                f_timdat ;       /* time & date stamp */
long                f_symptr ;       /* file ptr to symtab */
long                f_nsyms ;        /* # symtab entries */
unsigned short      f_opthdr ;       /* sizeof(opt hdr) */
unsigned short      f_flags ;        /* flags */
} ;
```

`f_symptr` is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in **fseek**(S) to position an I/O stream to the symbol table. The UNIX system optional header is 28 bytes. The valid magic numbers are given below:

```
#define I286SMAGIC      0512 /* 80286 computers-small model programs */
#define I286LMAGIC      0522 /* 80286 computers-large model programs */
#define I386MAGIC       0514 /* 80386 computers */
#define FBOMAGIC        0560 /* 3B2 and 3B15 computers */
#define N3BMAGIC        0550 /* 3B20 computer */
#define NTVMAGIC        0551 /* 3B20 computer */
#define VAXWRMAGIC      0570 /* VAX writable text segments */
#define VAXROMAGIC      0575 /* VAX read only sharable */
```

The value in `f_timdat` is obtained from the **time**(S) system call. Flag bits currently defined are:

```
#define F_RELFLG        0000001 /* relocation entries stripped */
#define F_EXEC          0000002 /* file is executable */
#define F_LNNO          0000004 /* line numbers stripped */
#define F_LSYMS         0000010 /* local symbols stripped */
#define F_MINMAL        0000020 /* minimal object file */
#define F_UPDATE        0000040 /* update file, ogen produced */
#define F_SWABD         0000100 /* file is "pre-swabbed" */
#define F_AR16WR        0000200 /* 16-bit DEC host */
#define F_AR32WR        0000400 /* 32-bit DEC host */
#define F_AR32W         0001000 /* non-DEC host */
#define F_PATCH         0002000 /* "patch" list in opt hdr */
#define F_80186         010000  /* contains 80186 instructions */
#define F_80286         020000  /* contains 80286 instructions */
#define F_BM32ID        0160000 /* WE32000 family ID field */
#define F_BM32B         0020000 /* file contains WE 32100 code */
#define F_BM32MAU       0040000 /* file reqs MAU to execute */
#define F_BM32RST       0010000 /* this object file contains restore */
```

See also

a.out(FP), **fseek**(S), **time**(S)

filesystem

format of filesystem types

Syntax

```
#include <sys/fs/*filsys.h>
#include <sys/types.h>
#include <sys/param.h>
```

Description

Every filesystem storage volume (for example, a hard disk) has a common format for certain vital information. Every such volume is divided into a certain number of 1024-byte blocks. There are four filesystem types available:

S51K (UNIX filesystem)
XENIX
AFS (ACER Fast Filesystem)
DOS

The DOS filesystem is a 512-byte filesystem. (The DOS filesystem structure is shown in */usr/include/fs/dosfilsys.h*. Consult a DOS reference guide for more information.)

Block 0 is unused and available to contain a bootstrap program or other information.

Block 1 is the "super block." The formats of the S51K, AFS, and XENIX filesystem super blocks are described in two files in the directory */usr/include/sys/fs*: *s5filsys.h* (S51K and AFS) and *xxfilsys.h* (XENIX). The XENIX filesystem boot block is 1024 bytes; the S51K and UNIX boot blocks are 512-byte blocks. In these include files, **s_isize** is the address of the first data block after the i-list. The i-list starts just after the super block in block 2; thus the i-list is **s_isize**-2 blocks long. **s_fsize** is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers. If an "impossible" block number is allocated from the free list or is freed, a diagnostic is written on the console. Moreover, the free array is cleared so as to prevent further allocation from a presumably corrupted free list.

The free list for S51K and XENIX volumes (but not AFS) is maintained as follows: The **s_free** array contains, in **s_free**[1], ..., **s_free**[**s_nfree**-1], up to **NICFREE**-1 numbers of free blocks. **s_free**[0] is the block number of the head of a chain of blocks constituting the free list. The first short in each free-chain block is the number (up to **NICFREE**) of free-block numbers listed in the next **NICFREE** longs of this chain member. The first of these **NICFREE** blocks is the link to the next member of the chain. To allocate a block: decrement **s_nfree**, and the new block is **s_free**[**s_nfree**]. If the new block number is 0, there are no blocks left, so give an error. If **s_nfree** becomes 0, read in the block named

by the new block number, replace **s_nfree** by its first word, and copy the block numbers in the next **NICFREE** longs into the **s_free** array. To free a block, check if **s_nfree** is **NICFREE**; if so, copy **s_nfree** and the **s_free** array into it, write it out, and set **s_nfree** to 0. In all cases, set **s_free[s_nfree]** to the freed block's number and increment **s_nfree**.

In the AFS filesystem, the free list is maintained differently The AFS free list is organized as a bitmap, one bit per (1K) block in the filesystem. This organization makes it easy to find contiguous stretches of free blocks.

s_tfree is the total free blocks available in the filesystem.

s_ninode is the number of free i-numbers in the **s_inode** array. To allocate an inode: if **s_ninode** is greater than 0, decrement it and return **s_inode[s_ninode]**. If it was 0, read the i-list and place the numbers of all free inodes (up to **NICINOD**) into the **s_inode** array, then try again. To free an inode, provided **s_ninode** is less than **NICINOD**, place its number into **s_inode[s_ninode]** and increment **s_ninode**. If **s_ninode** is already **NICINOD**, do not enter the freed inode into any table. This list of inodes only speeds up the allocation process. The information about whether the inode is free is maintained in the inode itself.

s_tinode is the total free inodes available in the file system.

The following applies only to S51K and AFS filesystems: **s_state** indicates the state of the filesystem. A cleanly unmounted, undamaged filesystem is indicated by the **FsOKAY** state. After a file system has been mounted for update, the state changes to **FsACTIVE**. A special case is used for the root file system. If the root file system appears damaged at boot time, it is mounted but marked **FsBAD**. Lastly, after a file system has been unmounted, the state reverts to **FsOKAY**.

s_flock and **s_ilock** are flags maintained in the core copy of the filesystem while it is mounted, and their values on disk are immaterial. The value of **s_fmod** on disk is also immaterial, and is used as a flag to indicate that the super block has changed and should be copied to the disk during the next periodic update of filesystem information.

s_ronly is a read-only flag to indicate write-protection.

s_time is the last time the super block of the file system was changed, and is a double-precision representation of the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the **s_time** of the super block for the root filesystem is used to set the system's time.

I-numbers begin at 1, and the storage for inodes begins in block 2. Inodes are 64 bytes long, so 16 of them fit into a block. Therefore, inode **i** is located in block (**i**+31)/16, and begins 64×((**i**+31) (mod 16)) bytes from its start. Inode 1 is reserved for future use. Inode 2 is reserved for the root directory of the filesystem, but no other i-number has a built-in meaning. Each inode represents one file. For the format of an inode and its flags, see **inode**(FP).

Files

/usr/include/sys/filsys.h

/usr/include/sys/stat.h

See also

fsck(ADM), **fsname**(ADM), **inode**(FP), **labelit**(ADM), **mkfs**(ADM)

inode

format of an inode

Syntax

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

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

Description

An inode for a plain file or directory in a file system has the structure defined by *<sys/ino.h>*. For the meaning of the defined types `off_t` and `time_t`, see *types*(FP).

Files

/usr/include/sys/ino.h

See also

filesystem(FP), **stat**(S), **types**(FP)

langinfo

language information constants

Syntax

```
#include <langinfo.h>
```

Description

This is a header file that contains constants used to identify items of **nl_langinfo**(S) data. (See **nl_langinfo**(S)).

The contents of the header file are:

```
/*
 *      LC_CTYPE   is not queried through this interface
 */

/*
 *      LC_NUMERIC        items:
 */

#define RADIXCHAR        2000        /* Decimal separator */
#define THOUSEP          2001        /* Thousands separator */

/*
 *      LC_TIME   items:
*/
#define D_FMT             3000        /* Date format only */
#define T_FMT             3001        /* Time format only */
#define PM_STR            3002        /* PM */
#define AM_STR            3003        /* AM */
#define D_T_FMT           3004        /* Date and time format */
#define DAY_1             3005        /* Sunday */
#define DAY_2             3006        /* Monday */
#define DAY_3             3007        /* Tuesday */
#define DAY_4             3008        /* Wednesday */
#define DAY_5             3009        /* Thursday */
#define DAY_6             3010        /* Friday */
#define DAY_7             3011        /* Saturday */
#define ABDAY_1           3012        /* Sun */
#define ABDAY_2           3013        /* Mon */
#define ABDAY_3           3014        /* Tue */
#define ABDAY_4           3015        /* Wed */
#define ABDAY_5           3016        /* Thu */
#define ABDAY_6           3017        /* Fri */
#define ABDAY_7           3018        /* Sat */
#define MON_1             3019        /* January */
#define MON_2             3020        /* February */
#define MON_3             3021        /* March */
#define MON_4             3022        /* April */
```

```
#define MON_5           3023        /* May */
#define MON_6           3024        /* June */
#define MON_7           3025        /* July */
#define MON_8           3026        /* August */
#define MON_9           3027        /* September */
#define MON_10          3028        /* October */
#define MON_11          3029        /* November */
#define MON_12          3030        /* December */
#define ABMON_1         3031        /* Jan */
#define ABMON_2         3032        /* Feb */
#define ABMON_3         3033        /* Mar */
#define ABMON_4         3034        /* Apr */
#define ABMON_5         3035        /* May */
#define ABMON_6         3036        /* Jun */
#define ABMON_7         3037        /* Jul */
#define ABMON_8         3038        /* Aug */
#define ABMON_9         3039        /* Sep */
#define ABMON_10        3040        /* Oct */
#define ABMON_11        3041        /* Nov */
#define ABMON_12        3042        /* Dec */

/*
 *  LC_COLLATE        is not queried through this interface
 */

/*
 *  LC_MESSAGES        items:
 */

#define YESSTR          5000  /* Affirmative reply to y/n question */
#define NOSTR           5001  /* Negative reply to y/n question */

/*
 *  LC_MONETARY               items:
 */

#define CRNCYSTR        6000  /* Currency symbol */
```

See also

nl_types(FP), **nl_langinfo**(S)

Standards conformance

langinfo is an extension of AT&T System V provided by The Santa Cruz Operation.

lconv

Internationalization (native language support) information

Syntax

```
#include <locale.h>
```

Description

The `lconv` structure contains the following members:

Member	Default	Description
decimal_point	"."	Decimal point
thousands_sep	""	Thousands separator
grouping	""	Size of groups of digits
int_curr_symbol	""	International currency symbol
currency_symbol	""	Currency symbol
mon_decimal_point	""	Monetary decimal point
mon_thousands_sep	""	Monetary thousands separator
mon_grouping	""	Monetary groupings
positive_sign	""	Positive sign
negative_sign	""	Negative sign
int_frac_digits frac_digits	CHAR_MAX CHAR_MAX	Number of digits used to display the fractions of a monetary unit; for example, two digits are used for $1.99
p_cs_precedes	CHAR_MAX	Whether the `currency_symbol` precedes (1) or succeeds (0) a positive monetary value; for example, $99.97 or .99$
p_sep_by_space	CHAR_MAX	Whether a space separates a `currency_symbol` from a positive value; for example, $99.97 (value is 0) or $ 99.97 (value is 1)
n_cs_precedes	CHAR_MAX	Whether a `currency_symbol` precedes (1) or succeeds (0) a negative monetary value; for example, $-

(Continued on next page)

(Continued)

Member	Default	Description
		99.97 or -.99$
n_sep_by_space	CHAR_MAX	Whether a space separates a `currency_symbol` from a negative value; for example, $-99.97 (value is 0) or $ -99.97 (value is 1)
p_sign_posn	CHAR_MAX	How to designate a positive monetary value. If the value is 0, parentheses are used, for example, ($99.97). If the value is 1, then the value of `positive_sign` is used; if 2 the sign string succeeds the value, if 3, the sign string precedes the value and the `currency_symbol`; if 4, the sign string immediately succeeds the `currency_symbol`. For example, if `positive_sign` is set to "+", then the result is +$99.97.
n_sign_posn	CHAR_MAX	How to designate a negative monetary value. Uses the same values as `p_sign_posn`, except that the sign string is the `negative_sign` string.

If a string member in the `lconv` structure has the value "" (the null string) or if a char member has the value CHAR_MAX, then the value for the member is is unavailable in the current locale.

CHAR_MAX is usually defined as decimal 127 unless the program is compiled with the -J compiler option. The -J option allows the user to treat char types as unsigned integers instead of signed integers. Thus, CHAR_MAX is changed in the header file *<limits.h>* to the value 255. However, the locale files were built with the assumption that CHAR_MAX equals 127. In this case you should use the value 127 explicitly instead of CHAR_MAX.

NOTE Currently, the only locale values supported for C are the default values listed above. Different values may be supported by other languages: these may be obtained from the appropriate locale documentation.

ldfcn

common object file access routines

Syntax

```
#include <stdio.h>

#include <filehdr.h>
#include <ldfcn.h>

#include INCDIR/filehdr.h
#include INCDIR/ldfcn.h
```

Description

The common object file access routines are a collection of functions for reading common object files and archives containing common object files. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type **LDFILE**, defined as `struct ldfile`, declared in the header file *ldfcn.h*. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function **ldopen**(S) allocates and initializes the **LDFILE** structure and returns a pointer to the structure to the calling program. The fields of the **LDFILE** structure may be accessed individually through macros defined in *ldfcn.h* and contain the following information:

LDFILE *ldptr;

TYPE(ldptr) The file magic number used to distinguish between archive members and simple object files

IOPTR(ldptr) The file pointer returned by **fopen** and used by the standard input/output functions

OFFSET(ldptr) The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.

HEADER(ldptr) The file header structure of the object file

The object file access functions themselves can be divided into the following categories:

- functions that open or close an object file:

  ldopen(S) and **ldaopen** open a common object file

  ldclose(S) and **ldaclose** close a common object file

- functions that read header or symbol table information:

  ldahread(S) reads the archive header of a member of an archive file

  ldfhread(S) reads the file header of a common object file

  ldshread(S) and **ldnshread** read a section header of a common object file

  ldtbread(S) reads a symbol table entry of a common object file

  ldgetname(S) retrieves a symbol name from a symbol table entry or from the string table

- functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section:

  ldohseek(S) seeks to the optional file header of a common object file

  ldsseek(S) and **ldnsseek** seek to a section of a common object file

  ldrseek(S) and **ldnrseek** seek to the relocation information for a section of a common object file

  ldlseek(S) and **ldnlseek** seek to the line number information for a section of a common object file

  ldtbseek(S) seeks to the symbol table of a common object file

- the function **ldtbindex**(S) which returns the index of a particular common object file symbol table entry.

These functions are described in detail on their respective manual pages. All the functions except **ldopen**(S), **ldgetname**(S), and **ldtbindex**(S) return either SUCCESS or FAILURE, both constants defined in *ldfcn.h*. **ldopen**(S) and **ldaopen** both return pointers to an LDFILE structure.

Additional access to an object file is provided through a set of macros defined in *ldfcn.h*. These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the LDFILE structure into a reference to its file descriptor field.

The following macros are provided:

GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)

The **STROFFSET** macro calculates the address of the string table. See the manual entries for the corresponding standard input/output library functions for details on the use of the rest of the macros.

The program must be loaded with the object file access library routine **LIBDIR/elibld.a**.

Notes

The macro **FSEEK** defined in the header file *ldfcn.h* translates into a call to the standard input/output function **fseek**(S). **FSEEK** should not be used to seek from the end of an archive file, since the end of an archive file may not be the same as the end of one of its object file members.

See also

fseek(S), **intro**(M) **ldahread**(S), **ldclose**(S), **ldfhread**(S), **ldgetname**(S), **ldlread**(S), **ldlseek**(S), **ldohseek**(S), **ldopen**(S), **ldrseek**(S), **ldshread**(S), **ldtbindex**(S), **ldtbread**(S), **ldtbseek**(S), **stdio**(S)

limits

header file for implementation-specific constants

Syntax

```
#include <limits.h>
```

Description

The header file *<limits.h>* is a list of magnitude limitations imposed by a specific implementation of the operating system. All values are specified in decimal.

```
#define ARG_MAX    5120                    /* max length of arguments to exec */
#define CHAR_BIT   8                       /* # of bits in a "char" */
#define CHAR_MAX   127                     /* max integer value of a "char" */
#define CHAR_MIN   -128                    /* min integer value of a "char" */
#define CHILD_MAX  25                      /* max # of processes per user id */
#define CLK_TCK    100                     /* # of clock ticks per second */
#define DBL_DIG    16                      /* digits of precision of a "double" */
#define DBL_MAX    1.7976931348623157e+308 /* "double" max decimal value */
#define DBL_MIN    2.2250738585072014e-308 /* "double" min decimal value */
#define FCHR_MAX   1048576                 /* max size of a file in bytes */
#define FLT_DIG    7                       /* digits of precision of a "float" */
#define FLT_MAX    ((float)3.40282347e+38  /* "float" max decimal value */
#define FLT_MIN    ((float)1.17549435e-38  /* "float" min decimal value */
#define HUGE_VAL   3.40282346638528860e+38 /* error val. returned by libm */
#define INT_MAX    2147483647              /* max decimal value of an "int" */
#define INT_MIN    -2147483648             /* min decimal value of an "int" */
#define LINK_MAX   1000                    /* max # of links to a single file */
#define LONG_MAX   2147483647              /* max decimal value of a "long" */
#define LONG_MIN   -2147483648             /* min decimal value of a "long" */
#define Name_MAX   14                      /* max # of characters in a filename */
#define OPEN_MAX   60                      /* max # of open files for a process */
#define PASS_MAX   8                       /* max # of characters in a password */
#define PATH_MAX   256                     /* max # of characters in a pathname */
#define PID_MAX    30000                   /* max value for a process ID */
#define PIPE_BUF   5120                    /* max # bytes atomic in write to a pipe */
#define PIPE_MAX   5120                    /* max # bytes written to a pipe */
#define SHRT_MAX   32767                   /* max decimal value of a "short" */
#define SHRT_MIN   -32768                  /* min decimal value of a "short" */
#define STD_BLK    1024                    /* # bytes in a physical I/O block */
#define SYS_NMLN   9                       /* # of chars in uname-returned strings */
#define UID_MAX    60000                   /* max value for a user or group ID */
#define USI_MAX    4294967295              /* max decimal value of an "unsigned" */
#define WORD_BIT   32                      /* # of bits in a "word" or "int" */
```

Standards conformance

limits is conformant with:
X/Open Portability Guide, Issue 3, 1989.

linenum

line number entries in a common object file

Syntax

```
#include <linenum.h>
```

Description

The cc command generates an entry in the object file for each C source line on which a breakpoint is possible (when invoked with the -g option; see cc(CP)). Users can then reference line numbers when using the appropriate software test system (see sdb(CP)). The structure of these line number entries follows:

```
struct     lineno
{
           union
           {
                   long      l_symndx ;
                   long      l_paddr ;
           } l_addr ;
           unsigned short    l_lnno ;
} ;
```

Numbering starts with one for each function. The initial line number entry for a function has `l_lnno` equal to zero, and the symbol table index of the function's entry is in `l_symndx`. Otherwise, `l_lnno` is non-zero, and `l_paddr` is the physical address of the code for the referenced line. Thus the overall structure is the following:

```
l_addr            l_lnno

function symtab   index        0
physical address  line
physical address  line
...

function symtab   index        0
physical address  line
physical address  line
...
```

See also

a.out(FP), **cc**(CP), **dbxtra**(CP), **codeview**(CP), **sdb**(CP)

mallinfo

malloc (libmalloc) information

Syntax

```
#include <malloc.h>
```

Description

The */usr/include/malloc.h* file defines the `mallinfo` structure, which provides information about:

Member	Description
arena	The total space in the allocated "arena" of memory
ordblks	Number of ordinary blocks
smblks	Number of small blocks
hblks	Number of holding blocks
hblkhd	Space in holding block headers
usmblks	Space in small blocks in use
fsmblks	Space in free small blocks
uordblks	Space in ordinary blocks in use
fordblks	Space in free ordinary blocks
keepcost	Cost of enabling keep option

mem, kmem

memory image file

Description

mem - provides access to the computer's physical memory

The *mem* file provides access to the computer's physical memory. All byte addresses in the file are interpreted as memory addresses. Thus, memory locations can be examined in the same way as individual bytes in a file. Note that accessing a nonexistent location causes an error.

kmem - provides access to the kernel virtual memory

The *kmem* file is the same as *mem* except that it corresponds to the kernel virtual memory rather than the computer's physical memory.

In rare cases, the *mem* and *kmem* files may be used to write to memory and memory-mapped devices. Such patching is not intended for the naive user and may lead to a system crash if not conducted properly. Patching device registers can lead to unexpected results if the device has read-only or write-only bits.

Files

/dev/mem

/dev/kmem

mfsys

configuration file for filesystem types

Description

The *mfsys* file contains configuration information for filesystem types that are to be included in the next system kernel to be built. It is included in the directory */etc/conf/cf.d*, and includes a one-line description of each filesystem type. The *mfsys* file is gathered from component files in the directory */etc/conf/mfsys.d*. Each line contains the following white space-separated fields:

- *name*: This field contains the internal name for the filesystem type (example: S51K, AFS). This name is no more than 32 characters long, and by convention is composed of uppercase alphanumeric characters.
- *prefix*: The name in this field is a string that is prefixed to the handler functions described in the `fstypsw` structure. The prefix name is up to 8 characters in length. The `fstypsw` structure is defined in the */usr/include/sys/conf.h* header file.
- *flags*: The *flags* field contains a hex number of the form "0xNN" to be used in populating the `fsinfo` data structure table entry for this filesystem type.
- *notify flags*: The *notify flags* field contains a hex number of the form "0xNN" to be used in populating the `fsinfo` data structure table entry for this filesystem type.
- *function bitstring*: The *function bitstring* is a string of 28 zeros and ones. Each filesystem type potentially defines 28 functions to populate the `fstypsw` data structure table entry for itself. All filesystem types do not supply all the functions in this table, however, and this bitstring is used to indicate which of the functions are present and which are absent. A "1" in this string indicates that a function has been supplied, and a "0" indicates that a function has not been supplied. Successive characters in the string represent successive elements of the `fstypsw` data structure, with the first entry in this data structure represented by the rightmost character in the string.

See also

idbuild(ADM), **idinstall**(ADM), **sfsys**(FP)

msg

IPC message structures

Syntax

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

Description

A message queue identifier (msqid) is a unique positive integer created by a **msgget**(S) system call. Each msqid has a message queue and a data structure associated with it. The data structure is referred to as **msqid_ds** and contains the following members:

```
struct  ipc_perm msg_perm;      /* operation permission struct */
struct  msg *msg_first;         /* ptr to first message on q */
struct  msg *msg_last;          /* ptr to last message on q */
ushort  msg_cbytes;             /* current number of bytes on q */
ushort  msg_qnum;               /* number of msgs on q */
ushort  msg_qbytes;             /* max number of bytes on q */
ushort  msg_lspid;              /* pid of last msgsnd operation */
ushort  msg_lrpid;              /* pid of last msgrcv operation */
time_t  msg_stime;              /* last msgsnd time */
time_t  msg_rtime;              /* last msgrcv time */
time_t  msg_ctime;              /* last change time */
                                /* Times measured in secs since */
                                /* 00:00:00 GMT, Jan. 1, 1970 */
```

msg_perm is an ipc_perm structure that specifies the message operation permission. The structure includes the following members:

```
ushort  uid;        /* owner's user id */
ushort  gid;        /* owner's group id */
ushort  cuid;       /* creator's user id */
ushort  cgid;       /* creator's group id */
ushort  mode;       /* r/w permission */
ushort  seq;        /* slot usage sequence number */
key_t   key;        /* key */
```

msg_qnum is the number of messages currently on the queue. msg_qbytes is the maximum number of bytes allowed on the queue. msg_lspid is the process ID of the last process that performed a **msgsnd** operation. msg_lrpid is the process ID of the last process that performed a **msgrcv** operation. msg_stime is the time of the last **msgsnd** operation, msg_rtime is the time of the last **msgrcv** operation, and msg_ctime is the time of the last **msgctl**(S) operation that changed a member in the preceding structure.

See also

msgctl(S), **msgget**(S), **msgop**(S)

netbuf

TLI/XTI network buffer structure

Syntax

```
#include <sys/tiuser.h>          /*  TLI only  */

#include <xti.h>                 /*  XTI only  */
```

Description

In the Network Services library, the **netbuf** structure is used in various Transport Layer Interface (TLI) functions to send and receive data and information. The **netbuf** structure contains the following members:

```
unsigned int maxlen;
unsigned int len;
char         *buf;
```

`buf` points to a user input and/or output buffer. `len` generally specifies the number of bytes contained in the buffer. If the structure is used for both input and output, the function replaces the user value of `len` on return.

`maxlen` generally has significance only when `buf` is used to receive output from the TLI function. In this case, it specifies the physical size of the buffer, the maximum value of `len` that can be set by the function. If `maxlen` is not large enough to hold the returned information, a **TBUFOVFLW** error generally results. However, certain functions may return part of the data and not generate an error.

See also

t_accept(S), **t_connect**(S), **t_getstate**(S), **t_listen**(S), **t_open**(S), **t_rcvconnect**(S)

nl_types

data types for native language support

Syntax

```
#include <nl_types.h>
```

Description

This is a header file that provides type definitions used in the native language support interface. The types are:

`nl_item` A type definition for an item of language data as used by **nl_langinfo**(S). The values for `nl_item` are defined in the file *<langinfo.h>*.

`nl_catd` message catalogue descriptor (Message catalogues are not currently supported.)

See also

langinfo(FP), **nl_langinfo**(S)

Standards conformance

nl_types is an extension of AT&T System V provided by The Santa Cruz Operation.

nssend

name server request structure

Syntax

```
#include <nserve.h>
```

Description

The **nssend** structure contains the information used to send requests to or receive requests from the name server.

The structure is:

```
short           ns_code;        /* request code - see list that follows */
short           ns_type;        /* type of data, unused for now        */
short           ns_flag;        /* read/write flag */
char            *ns_name;       /* name of resource */
char            *ns_desc;       /* description of resource */
char            *ns_path;       /* local pathname of resource */
struct address  *ns_addr;       /* address of resource's owner */
char            **ns_mach;      /* list of client machines */
```

The request codes for the `ns_code` member are as follows:

Code	Description
NS_ADV	Advertise a resource
NS_BYMACHINE	Inverse query by owner and type
NS_FINDP	Find which machine is primary
NS_GET	Unused query
NS_IM_NP	Indicate secondary poll
NS_IM_P	Indicate primary poll
NS_INIT	Called to set up name service
NS_IQUERY	General inverse query
NS_MODADV	Modify an advertisement
NS_QUERY	Query by name and type
NS_REL	Relinquish being primary
NS_SENDPASS	Register password with primary
NS_SNDBACK	Null type
NS_UNADV	Unadvertise a resource
NS_VERIFY	Verify validity of password

null

the null file

Description

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

Files

/dev/null

Standards conformance

null is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

passwd

password entry structure

Syntax

```
#include <pwd.h>
```

Description

The `passwd` structure contains the following fields for SVID:

Member	Description	
char *pw_name;	/* login name	*/
char *__pw_passwd;	/* password	*/
int pw_uid;	/* user ID	*/
int pw_gid;	/* group ID	*/
char *pw_age;	/* age of pw_passwd	*/
char *pw_comment;	/* user information	*/
char *pw_gecos;	/* historical field	*/
char *pw_dir;	/* user's login directory	*/
char *pw_shell;	/* user's login shell	*/

For XPG3 and POSIX the structure definition is as follows:

Member	Description	
char *pw_name;	/* login name	*/
char *pw_passwd;	/* password	*/
uid_t pw_uid;	/* user ID	*/
short __pad1;	/* padded space	*/
int pw_gid;	/* group ID	*/
short __pad2;	/* padded space	*/
char *__pw_age;	/* age of pw_passwd	*/
char *__pw_comment;	/* user information	*/
char *__pw_gecos;	/* historical field	*/
char *pw_dir;	/* user's login directory	*/
char *pw_shell;	/* user's login shell	*/

When neither XPG3 or POSIX is defined, the comment field can contain the following information defined in the `comment` structure:

Member	Description	
char *c_dept;	/* user's department	*/
char *c_name;	/* user's name	*/
char *c_acct;	/* user's account	*/
char *c_bin;	/* user's output storage bin ID	*/

The `pw_gecos` member is interpreted in the same way as the `pw_comment` member. The acronym "gecos" stands for General Electric Comprehensive Operating System. The `pw_gecos` member is used only as a comment.

Required fields

The SVID (Issue 2) **passwd**(BA_ENV) routine specifies that only the encrypted password, GID, and shell fields can be empty. XPG2 **passwd**(4) is identical. Both standards specify that the comment field is implementation specific, as does this implementation of the Development System. Neither POSIX nor XPG3 specifies file formats.

In summary:

Member	SVID-2/ XPG2	SCO XENIX 2.2	SCO XENIX 2.3	SCO UNIX
pw_name	r	e	e	e
pw_passwd	e	e	e	e
pw_uid	r	e	r	r
pw_gid	e	e	r	r
pw_gecos	e	e	e	e
pw_dir	r	e	e	e
pw_shell	e	e	e	e
(rest)	i	i	i	i

Where:

r	= required
e	= may be empty
i	= ignored
(rest)	= Any fields after the login shell pathname

None of the standards states whether spaces are allowed in front of the UID or GID, but both SCO UNIX and XENIX 2.3 allow spaces.

/etc/passwd entries

The information in the `passwd` structure is used to create an entry in the */etc/passwd* that is parsed with the **fgetpwent**(S) routine. Each member in the structure in which data is present is placed in the password file entry separated by colons, as shown in this example:

```
rufusf:*:203:50:Rufus T. Firefly:/usr/rufusf:/bin/sh
```

In this case, the members shown are, respectively, `pw_name`, `pw_passwd`, `pw_uid`, `pw_gid`, `pw_comment`, `pw_dir`, and `pw_shell`. When an asterisk is present in the `pw_passwd` member, the password is handled by a security scheme. Refer to the *System Administrator's Guide* for more information on password security.

The full length of an */etc/passwd* entry cannot exceed **BUFSIZ** characters.

plot

graphics interface

Description

Files of this format are produced by routines described in **plot**(S) and are interpreted for various devices by commands described in **tplot**(ADM). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the *x* and *y* values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in **plot**(S).

m move: The next four bytes generate a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes (see **tplot**(ADM)).

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline character.

e erase: Start another frame of output.

f linemod: Take the following string, up to a new line, as the style for drawing further lines. The styles are **dotted**, **solid**, **longdashed**, **shortdashed**, and **dotdashed**. Effective only for the **-T4014** and **-Tver** options of **tplot**(ADM) (TEKTRONIX 4014 terminal and VERSATEC plotter).

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot is magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unit-y scaling are shown for devices supported by the filters of **tplot**(ADM). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);
DASI 450	space(0, 0, 4096, 4096);
TEKTRONIX 4014	space(0, 0, 3120, 3120);
VERSATEC plotter	space(0, 0, 2048, 2048);

Note

The plotting library **plot**(S) and the curses library **curses**(S) both use the names **erase** and **move**. The **curses** versions are macros. If you need both libraries, put the **plot**(S) code in a different source file than the **curses**(S) code, and/or undefine (using **#undef move** and **erase** in the **plot**(S) code.

See also

graph(ADM), **plot**(S), **term**(M), **tplot**(ADM)

pnch

file format for card images

Description

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0 to 80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.

proc

process table structure

Syntax

```
#include  <sys/types.h>
#include  <sys/proc.h>

short p_pgrp;   /* process group */
short p_pid;    /* process ID */
pid_t p_sid;    /* POSIX session ID number */
```

Description

This structure contains information about a process and its state in the kernel. Even though many fields exist, only `p_pgrp`, `p_pid`, and `p_sid` fields can be used in a device driver. All fields in the **proc** structure are for reading only; do not write to any field in this structure. The process table can only be read in the task-time section of a device driver; that is, when a driver has user context.

The `p_sid` field is related to the `p_pgrp` field in its use and value. In most cases, these fields have the same value, but under special circumstances determined by a driver's implementation, they can differ.

If the `p_pgrp` and the `p_pid` fields contain the same value, then the process is the process group leader.

The `u.u_procp` field in the **user** structure contains a pointer to the **proc** structure.

The **setpgrp**(S) and **setsid**(S) system calls provide direct writing capability to the **proc** structure. These are the only means that should be used to write to this structure.

See also

setpgrp(S), **setsid**(S)

reloc

relocation information for a common object file

Syntax

```
#include <reloc.h>
```

Description

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format.

```
struct  reloc
{
        long        r_vaddr ;       /* (virtual) address of reference */
        long        r_symndx ;      /* index into symbol table */
        short       r_type ;        /* relocation type */
} ;

# define R_PCRLONG  024
```

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct how references found within the input section are treated.

R_PCRLONG A "PC-relative" 32-bit reference to the symbol's virtual address.

More relocation types exist for other processors. Equivalent relocation types on different processors have equal values and meanings. New relocation types will be defined (with new values) as they are needed.

Relocation entries are generated automatically by the assembler and automatically used by the link editor. Link editor options exist for both preserving and removing the relocation entries from object files.

See also

a.out(FP), **as**(CP), **ld**(CP), **syms**(FP)

sccsfile

format of an SCCS file

Description

An SCCS file is an ASCII file. It consists of six logical parts: the ***checksum***, the ***delta table*** (contains information about each delta), ***user names*** (contains log-in names and/or numerical group IDs of users who may add deltas), ***flags*** (contains definitions of internal keywords), ***comments*** (contains arbitrary descriptive information about the file), and the ***body*** (contains the actual text lines intermixed with control lines).

Throughout an SCCS file, there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and will be represented graphically as @. Any line described in this description which is not depicted as beginning with the control character is prevented from beginning with the control character. Entries of the form DDDDD represent a five-digit string (a number between 00000 and 99999).

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

```
@hDDDDD
```

The value of the checksum is the sum of all characters, except those of the first line. The @hR provides a *magic number* of (octal) 064001.

Delta table

The delta table consists of a variable number of entries of the form:

```
@s DDDDD /DDDDD /DDDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
 .
 .
@c <comments> ...
 .
 .
@e
```

The first line (@s) contains the number of lines inserted/deleted/unchanged respectively. The second line (@d) contains the type of the delta (currently, normal: D, and removed: R), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the

time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The @i, @x, and @g lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

Each @m line (optional) contains one MR number associated with the delta; the @c lines contain comments associated with the delta.

The @e line ends the delta table entry.

User names

This list of login names and/or numerical group IDs includes of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta.

Flags

Flags are keywords used internally (see **admin**(CP) for more information on their use). Each flag line takes the form:

```
@f <flag>        <optional text>
```

The following flags are defined:

```
@f t    <type of program>
@f v    <program name>
@f i
@f b
@f m    <module name>
@f f    <floor>
@f c    <ceiling>
@f d    <default-sid>
@f n
@f j
@f l    <lock-releases>
@f q    <user defined>
```

The **t** flag defines the replacement for the identification keyword. The **v** flag controls prompting for MR numbers in addition to comments; if the optional text is present, it defines an MR number validity checking program. The **i** flag controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a "fatal" error (the file will not be received, or the delta will not be made). When the **b** flag is present the **-b** option may be used with the **get** command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the *sccsfile.F* identification keyword. The **f** flag defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release

above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a **get** command. The **n** flag causes delta to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes **get** to allow concurrent edits of the same base SID. The **l** flag defines a list of releases that are locked against editing (**get**(CP) with the **-e** option). The **q** flag defines the replacement for the identification keyword.

Comments

Comments are arbitrary text surrounded by the bracketing lines @t and @T. The comments section typically contains a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, as follows:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

The digit string DDDDD is the serial number corresponding to the delta for the control line.

See also

admin(CP), **delta**(CP), **get**(CP), **prs**(CP)

Standards conformance

sccsfile is conformant with:

AT&T SVID Issue 2.

scnhdr

section header for a common object file

Syntax

```
#include <scnhdr.h>
```

Description

Every common object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below.

```
struct  scnhdr
{
        char            s_name[SYMNMLEN];       /* section name */
        long            s_paddr;                /* physical address */
        long            s_vaddr;                /* virtual address */
        long            s_size;                 /* section size */
        long            s_scnptr;               /* file ptr to raw data */
        long            s_relptr;               /* file ptr to relocation */
        long            s_lnnoptr;              /* file ptr to line numbers */
        unsigned short  s_nreloc;               /* # reloc entries */
        unsigned short  s_nlnno;                /* # line number entries */
        long            s_flags;                /* flags */
};
```

File pointers are byte offsets into the file; they can be used as the offset in a call to **fseek** (see **ldfcn**(FP)). If a section is initialized, the file contains the actual bytes. An uninitialized section is somewhat different. It has a size, symbols defined in it, and symbols that refer to it. But it can have no relocation entries, line numbers, or data. Consequently, an uninitialized section has no raw data in the object file, and the values for `s_scnptr`, `s_relptr`, `s_lnnoptr`, `s_nreloc`, and `s_nlnno` are zero.

See also

a.out(FP), **fseek**(S), **ld**(CP)

scr_dump

format of curses screen image file

Syntax

```
scr_dump(file)
```

Description

The **curses**(S) function **scr_dump()** copies the contents of the screen into a file. The format of the screen image is described in this section.

The name of the tty is 20 characters long and the modification time is of the type **time_t**. All other numbers and characters are stored as **chtype** (see <*curses.h*>). No new lines are stored between fields.

```
<magic number: octal 0433>
<name of tty>
<mod time of tty>
<columns> <lines>
<line length> <chars in line>          for each line on the screen
<line length> <chars in line>
  .
  .
  .
<labels?>          1, if soft screen labels are present
<cursor row> <cursor column>
```

Only as many characters as are in a line are listed. For example, if the "line length" is "0", there are no characters following "line length". If "labels?" is "TRUE", following it is:

```
<number of labels>
<label width>
<chars in label 1>
<chars in label 2>
  .
  .
  .
```

See also

curses(S)

sem

controls structures for UNIX System V semaphores

Syntax

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

Description

A semaphore identifier (*semid*) is a unique positive integer created by a **semget**(S) system call. Each *semid* has a set of semaphores and a data structure associated with it. The data structure is referred to as **semid_ds** and contains the following members:

```
struct  ipc_perm sem_perm;      /* operation permission struct */
struct  sem *sem_base;          /* ptr to first semaphore in set */
ushort  sem_nsems;              /* number of sems in set; see below */
struct  lockb sem_cilock;       /* lock for MPX */
time_t  sem_otime;              /* time of last semop (S) operation */
time_t  sem_ctime;              /* time of last semctl (S) change */
                                /* Times measured in secs since */
                                /* 00:00:00 GMT, Jan. 1, 1970 */
```

sem_perm is an **ipc_perm** structure that specifies the semaphore operation permission. This structure includes the following members:

```
ushort  uid;        /* user id */
ushort  gid;        /* group id */
ushort  cuid;       /* creator user id */
ushort  cgid;       /* creator group id */
ushort  mode;       /* access permission (see below) */
ushort  seq;        /* slot usage sequence number */
key_t   key;        /* key */
```

The value of sem_nsems is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a positive integer referred to as a **sem_num**. **sem_num** values run sequentially from 0 to the value of sem_nsems minus 1. sem_otime is the time of the last **semop**(S) operation, and sem_ctime is the time of the last **semctl**(S) operation that changed a member of the above structure.

A semaphore is a data structure that contains the following members:

```
ushort  semval;         /* semaphore value */
short   sempid;         /* pid of last operation  */
ushort  semncnt;        /* # awaiting semval > cval */
ushort  semzcnt;        /* # awaiting semval = 0 */
```

`semval` is a non-negative unsigned short. `sempid` is equal to the process ID of the last process that performed a semaphore operation on this semaphore. `semncnt` is a count of the number of processes that are currently suspended waiting for this semaphore's `semval` to become greater than its current value. `semzcnt` is a count of the number of processes that are currently suspended waiting for this semaphore's `semval` to become zero.

Upon creation, the data structure associated with the new semaphore identifier is initialized as follows:

- `sem_perm.cuid`, `sem_perm.uid`, `sem_per.cgid`, and `sem_perm.gid` are set equal to the effective user ID and effective group ID of the calling process.
- The low-order 9 bits of `sem_perm.mode` are set equal to the low-order 9 bits of the *semflg* argument.
- `sem_nsems` is set equal to the value of **nsems**.
- `sem_otime` is set equal to 0 and `sem_ctime` is set equal to the current time.

The program that creates the semaphore array must issue the **semctl**(S) call with the SETVAL command to establish the initial value of the `sem` structure.

In addition to the structures described here, each process using the semaphore array must create a local data structure, identified by the *sops* argument to the **semop** call. See the **semop**(S) reference page for a description of the members of that structure.

Semaphore operation permissions

Access permissions for the semaphore are set according to values given to the *semflg* argument to the **semget**(S) system call, defined as:

00400	Read by user (SEM_R)
00200	Alter by user (SEM_W)
00040	Read by group
00020	Read by others
00004	Alter by group
00002	Alter by others

Read and alter permissions for a *semid* are granted to a process if one or more of the following is true:

- The effective user ID of the process is super user.
- The effective user ID of the process matches `sem_perm.uid` or `sem_perm.cuid` in the data structure associated with *semid* and the appropriate "user" portion (0600) bit of `sem_perm.mode` is set.

- The effective group ID of the process matches `sem_perm.gid` or `sem_perm.cgid` in the data structure associated with ***semid*** and the appropriate bit of the "user" portion (060) of `sem_perm.mode` is set.
- The appropriate bit of the "other" portion (06) of `sem_perm.mode` is set.

Otherwise, the corresponding permissions are denied.

Note that permission 7 is equivalent to permission 6 (thus, 0660 and 0770 are equivalent).

See also

ipcrm(ADM), **semctl**(S), **semget**(S), **semop**(S)

sfsys

local filesystem type file

Syntax

```
/etc/conf/cf.d/sfsys
```

Description

The *sfsys* file contains local system information about each file system type specified in the *mfsys* file. It is present in the directory */etc/conf/cf.d*, and contains a one-line entry for each file system type specified in the *mfsys* file. The *sfsys* file is coalesced from component files in the directory */etc/conf/sfsys.d*. Each line in this file is a whitespace-separated set of fields that specifies:

name: This field contains the internal name of the filesystem type (for example, DUFST, S51K). By convention, this name is up to 32 characters long, and is composed of all uppercase alphanumeric characters.

Y/N: This field contains either an uppercase "Y" (for "yes") or an uppercase "N" (for "no") to indicate whether the named file system type is to be configured into the next system kernel to be built.

See also

idinstall(ADM), **idbuild**(ADM), **mfsys**(FP)

shm

IPC shared memory structures

Syntax

```
/etc/conf/cf.d/sfsys
```

Description

A shared memory identifier (*shmid*) is a unique positive integer created by a **shmget**(S) system call. Each shmid has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. The data structure is referred to as **shmid_ds** and contains the following members:

```
struct  ipc_perm shm_perm;      /* operation permission struct */
int     shm_segsz;              /* size of segment */
ushort  shm_cpid;               /* creator pid */
ushort  shm_lpid;               /* pid of last operation */
short   shm_nattch;             /* number of current attaches */
time_t  shm_atime;              /* last attach time */
time_t  shm_dtime;              /* last detach time */
time_t  shm_ctime;              /* last change time */
                                /* Times measured in secs since */
                                /* 00:00:00 GMT, Jan. 1, 1970 */
```

shm_perm is an **ipc_perm** structure that specifies the shared memory operation permission. The structure includes the following members:

```
ushort  cuid;                   /* creator user id */
ushort  cgid;                   /* creator group id */
ushort  uid;                    /* user id */
ushort  gid;                    /* group id */
ushort  mode;                   /* r/w permission */
```

shm_segsz specifies the size of the shared memory segment. shm_cpid is the process ID of the process that created the shared memory identifier. shm_lpid is the process ID of the last process that performed a **shmop**(S) operation. shm_nattch is the number of processes that currently have this segment attached. shm_atime is the time of the last **shmat** operation. shm_dtime is the time of the last **shmdt** operation, and shm_ctime is the time of the last **shmctl**(S) operation that changed one of the above structure members.

In the **shmop**(S) and **shmctl**(S) system call descriptions, the permission required for an operation is given as "{token"}, where "token" is the type of permission needed. It is interpreted as follows:

00400	Read by user
00200	Write by user
00060	Read, write by group
00006	Read, write by others

Read and write permissions on a ***shmid*** are granted to a process if one or more of the following are true:

- The effective user ID of the process is super user.
- The effective user ID of the process matches `shm_perm.uid` or `shm_perm.cuid` in the data structure associated with ***shmid*** and the appropriate bit of the "user" portion (0600) of `shm_perm.mode` is set.
- The effective user ID of the process does not match `shm_perm.uid` or `shm_perm.cuid` and the effective group ID. The process matches `shm_perm.gid` or `shm_perm.cgid` and the appropriate bit of the "group" portion (060) of `shm_perm.mode` is set.
- The effective user ID of the process does not match `shm_perm.uid` or `shm_perm.cuid` and the effective group ID of the process does not match `shm_perm.gid` or `shm_perm.cgid` and the appropriate bit of the "other" portion (06) of `shm_perm.mode` is set.

Otherwise, the corresponding permissions are denied.

See also

shmctl(S), **shmget**(S), **shmop**(S)

stat

data returned by stat system call

Syntax

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

Description

The *sys/stat.h* include file contains the definition for the structure returned by the **stat** and **fstat** functions. The structure is defined as:

```
dev_t    st_dev;        /* ID of device containing */
                        /* a directory entry for this file */
ushort   st_ino;        /* Inode number */
ushort   st_mode;       /* File mode (see mknod (S)) */
short    st_nlink;      /* Number of links */
ushort   st_uid;        /* User ID of the file's owner */
ushort   st_gid;        /* Group ID of the file's group */
dev_t    st_rdev;       /* ID of device */
                        /* This entry is defined only for */
                        /* character special or */
                        /* block special files */
off_t    st_size;       /* File size in bytes */
time_t   st_atime;      /* Time of last access */
time_t   st_mtime;      /* Time of last data modification */
time_t   st_ctime;      /* Time of last file status change */
                        /* Times measured in seconds since */
                        /* 00:00:00 GMT, Jan. 1, 1970 */
```

The `st_mode` value is actually a combination of one or more of the following file mode values:

```
S_IFMT     0170000      /* type of file */
S_IFREG    0100000      /* regular */
S_IFBLK    0060000      /* block special */
S_IFNAM    0050000      /* special named file */
S_IFDIR    0040000      /* directory */
S_IFCHR    0020000      /* character special */
S_IFIFO    0010000      /* fifo */
```

The following definitions provide POSIX compliance of file permissions:

```
S_IRWXU    00700        /* read, write, execute: owner */
S_IRUSR    00400        /* read permission: owner */
S_IREAD    00400        /* read permission, owner */
S_IWUSR    00200        /* write permission: owner */
S_IWRITE   00200        /* write permission, owner */
S_IXUSR    00100        /* execute permission: owner */
S_IEXEC    00100        /* execute/search permission, owner */
S_IRWXG    00070        /* read, write, execute: group */
S_IRGRP    00040        /* read permission: group */
S_IWGRP    00020        /* write permission: group */
S_IXGRP    00010        /* execute permission: group */
S_IRWXO    00007        /* read, write, execute: other */
S_IROTH    00004        /* read permission: other */
S_IWOTH    00002        /* write permission: other */
S_IXOTH    00001        /* execute permission: other */
S_ISUID    04000        /* set user id on execution */
S_ISGID    02000        /* set group id on execution */
S_ISVTX    01000        /* save swapped text even after use */
```

Files

/usr/include/sys/stat.h

See also

stat(S)

Standards conformance

stat is conformant with:

AT&T SVID Issue 2;
X/Open Portability Guide, Issue 3, 1989;
IEEE POSIX Std 1003.1-1990 System Application Program Interface (API) [C Language] (ISO/IEC 9945-1);
and NIST FIPS 151-1.

syms

common object file symbol table format

Syntax

```
#include <syms.h>
```

Description

Common object files contain information to support symbolic software testing (see **sdb**(CP)). Line number entries, **linenum**(FP), and extensive symbolic information permit testing at the C *source* level. Every object file's symbol table is organized as shown below.

File name 1.
    Function 1.
        Local symbols for function 1.
    Function 2.
        Local symbols for function 2.
    ...
    Static externs for file 1.

File name 2.
    Function 1.
        Local symbols for function 1.
    Function 2.
        Local symbols for function 2.
    ...
    Static externs for file 2.
...

Defined global symbols.
Undefined global symbols.

The entry for a symbol is a fixed-length structure. The members of the structure hold the name (null padded), its value, and other information. The C structure follows:

```
#define  SYMNMLEN      8
#define  FILNMLEN      14
#define  DIMNUM        4

struct  syment
{
    union                                   /* all ways to get symbol name */
    {
        char        _n_name[SYMNMLEN];      /* symbol name */
        struct
        {
            long    _n_zeroes;              /* == 0L when in string table */
            long    _n_offset;              /* location of name in table */
        } _n_n;
        char        *_n_nptr[2];            /* allows overlaying */
    } _n;
    long            n_value;                /* value of symbol */
    short           n_scnum;                /* section number */
    unsigned short  n_type;                 /* type and derived type */
    char            n_sclass;               /* storage class */
    char            n_numaux;               /* number of aux entries */
};

#define  n_name     _n._n_name
#define  n_zeroes   _n._n_n._n_zeroes
#define  n_offset   _n._n_n._n_offset
#define  n_nptr     _n._n_nptr[1]
```

Meaningful values and their explanations are given in both **syms.h** and Common Object File Format (COFF). To interpret the entries see these sources. Some symbols require more information than a single entry; they are followed by auxiliary entries that are the same size as a symbol entry. The format follows.

```
union auxent
{
    struct
    {
        long        x_tagndx;
        union
        {
            struct
            {
                unsigned short  x_lnno;
                unsigned short  x_size;
            } x_lnsz;
            long    x_fsize;
        } x_misc;
        union
        {
            struct
            {
                long    x_lnnoptr;
                long    x_endndx;
            }   x_fcn;
            struct
            {
                unsigned short  x_dimen[DIMNUM];
            }   x_ary;
        }   x_fcnary;
        unsigned short  x_tvndx;
    }   x_sym;

    union
    {
        char    x_fname[FILNMLEN];
        struct
        {
            long x_zero;
            long x_offset;
        }   x_longname;
    }   x_file;

    struct
    {
        long    x_scnlen;
        unsigned short  x_nreloc;
        unsigned short  x_nlinno;
    }   x_scn;

    struct
    {
        long            x_tvfill;
        unsigned short  x_tvlen;
        unsigned short  x_tvran[2];
    }   x_tv;
};
```

Indexes of symbol table entries begin at zero.

Notes

On machines on which **ints** are equivalent to **longs**, all **longs** have their type changed to **int.** Thus the information about which symbols are declared as **longs** and which as **ints** does not show up in the symbol table.

See also

a.out(FP), **linenum**(FP), **sdb**(CP)

t_info

TLI and XTI transport protocol structure

Syntax

```
#include <sys/tiuser>                /* TLI only */

#include <xti.h>   /* XTI only */
```

Description

The `t_info` structure contains the following members:

Member	Description
long addr;	/* max size of the transport protocol address */
long options;	/* max number of bytes of protocol-specific options */
long tsdu;	/* max size of a transport service data unit (TSDU) */
long etsdu;	/* max size of expedited transport service data */ /* unit (ETSDU) */
long connect;	/* max data allowed on connection establishment */
long discon;	/* max data allowed for **t_snddis** and **t_rcvdis** */
long servtype;	/* service type supported by the transport provider */

The values of the members have the following meanings:

`addr` A value greater than or equal to zero indicates the maximum size of a transport protocol address; a value of -1 specifies that there is no limit on the address size; and a value of -2 specifies that the transport provider does not provide user access to transport protocol addresses.

`options` A value greater than or equal to zero indicates the maximum number of bytes of protocol-specific options supported by the provider; a value of -1 specifies that there is no limit on the option size; and a value of -2 specifies that the transport provider does not support user-settable options.

`tsdu` A value greater than zero specifies the maximum size of a transport service data unit (TSDU); a value of zero specifies that the transport provider does not support the concept of TSDU, although it does support the sending of a data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of a TSDU; and a value of -2 specifies that the transfer of normal data is not supported by the transport provider.

`etsdu` A value greater than zero specifies the maximum size of an Expedited Transport Service Data Unit (ETSDU); a value of zero specifies that the transport provider does not support the concept

of ETSDU, although it does support the sending of an expedited data stream with no logical boundaries preserved across a connection; a value of -1 specifies that there is no limit on the size of an ETSDU; and a value of -2 specifies that the transfer of expedited data is not supported by the transport provider.

`connect` A value greater than or equal to zero specifies the maximum amount of data that can be associated with connection establishment functions; a value of -1 specifies that there is no limit on the amount of data sent during connection establishment; and a value of -2 specifies that the transport provider does not allow data to be sent with connection establishment functions.

`discon` A value greater than or equal to zero specifies the maximum amount of data that can be associated with the **t_snddis**(S) and **t_rcvdis**(S) functions; a value of -1 specifies that there is no limit on the amount of data sent with these abortive release functions; and a value of -2 specifies that the transport provider does not allow data to be sent with the abortive release functions.

`servtype` This member specifies the service type supported by the transport provider, as described below.

If a transport user is concerned with protocol independence, the above sizes can be accessed to determine how large the buffers must be to hold each piece of information. Alternatively, the **t_alloc**(S) function can be used to allocate these buffers. An error results if a transport user exceeds the allowed data size on any function. The value of each member can change as a result of option negotiation, and **t_getinfo**(S) enables a user to retrieve the current characteristics.

The `servtype` member of `t_info` structure may specify one of the following values on return:

T_COTS The transport provider supports a connection-mode service but does not support the optional orderly release facility.

T_COTS_ORD The transport provider supports a connection-mode service with the optional orderly release facility.

T_CLTS The transport provider supports a connectionless-mode service. For this service type, **t_open**(S) returns -2 for `etsdu`, `connect`, and `discon`.

types

primitive system data types

Syntax

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

Description

The data types defined in the include file *<sys/types.h>* are used in UNIX system code; some of these data types are accessible to user code.

The form **daddr_t** is used for disk addresses except in an inode on disk (see *filesystem*(FP)). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The **label_t** variables are used to save the processor state while another process is running.

See also

filesystem(FP)

Standards conformance

types is conformant with:

X/Open Portability Guide, Issue 3, 1989.

unistd

file header for symbolic constants

Syntax

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

Description

The header file *<unistd.h>* lists the symbolic constants and structures not already defined or declared in some other header file.

```
/* Symbolic constants for the "access" routine: */

#define R_OK        4       /* Test for Read permission */
#define W_OK        2       /* Test for Write permission */
#define X_OK        1       /* Test for Execute permission */
#define F_OK        0       /* Test for existence of File */

#define F_ULOCK     0       /* Unlock a previously locked region */
#define F_LOCK      1       /* Lock a region for exclusive use */
#define F_TLOCK     2       /* Test and lock a region for exclusive use */
#define F_TEST      3       /* Test a region for other processes locks */

/* Symbolic constants for the "lseek" routine: */

#define SEEK_SET    0       /* Set file pointer to "offset" */
#define SEEK_CUR    1       /* Set file pointer to current plus "offset" */
#define SEEK_END    2       /* Set file pointer to EOF plus "offset" */

/* Path names: */

#define GF_PATH     "/etc/group"        /* Path name of the group file */
#define PF_PATH     "/etc/passwd"        /* Path name of the passwd file */
```

Standards conformance

unistd is conformant with:

X/Open Portability Guide, Issue 3, 1989.

x.out

format of XENIX link editor output

Syntax

```
#include <x.out.h>
```

Description

The output of the XENIX link editor, called the *x.out* or segmented *x.out* format, is defined by the files */usr/include/x.out.h* and */usr/include/sys/relsym.h*. The *x.out* file has the following general layout:

- header
- extended header
- file segment table (for segmented formats)
- segments (Text, Data, Symbol, and Relocation)

In the segmented format, there may be several text and data segments, depending on the memory model of the program. Segments within the file begin on boundaries that are multiples of 512 bytes, as defined by the file's page size.

Format

```
/*
 *The main and extended header structures.
 *For x.out segmented (XE_SEG):
 *    1) fields marked with (s) must contain sums of xs_psize for
 *        non-memory images, or xs_vsize for memory images.
 *    2) the contents of fields marked with (u) are undefined.
 */

struct xexec {                          /* x.out header */
        unsigned short    x_magic       /* magic number */
        unsigned short    x_ext;        /* size of header extension */
        long              x_text;       /* size of text segment (s) */
        long              x_data;       /* size of initialized data (s) */
        long              x_bss;        /* size of uninitialized data (s) */
        long              x_syms;       /* size of symbol table (s) */
        long              x_reloc       /* relocation table length (s) */
        long              x_entry       /* entry point, machine dependent */
        char              x_cpu;        /* cpu type & byte/word order */
        char              x_relsy;      /* relocation & symbol format (u) */
        unsigned short    x_renv;       /* run-time environment */
};
```

```
struct xext {                   /* x.out header extension */
        long            xe_trsize;      /* size of text relocation (s) */
        long            xe_drsize;      /* size of data relocation (s) */
        long            xe_tbase;       /* text relocation base (u) */
        long            xe_dbase;       /* data relocation base (u) */
        long            xe_stksize;     /* stack size (if XE_PS set) */
                /* the following must be present if XE_SEG */
        long            xe_segpos;      /* segment table position */
        long            xe_segsize;     /* segment table size */
        long            xe_mdtpos;      /* machine dependent table position */
        long            xe_mdtsize;     /* machine dependent table size */
        char            xe_mdttype;     /* machine dependent table type */
        char            xe_pagesize;    /* file page size, in multiples of 512 */
        char            xe_ostype;      /* operating system type */
        char            xe_osvers;      /* operating system version */
        unsigned short  xe_eseg;        /* entry segment, machine dependent */
        unsigned short  xe_sres;        /* reserved */
};

struct xseg {                   /* x.out segment table entry */
        unsigned short  xs_type;        /* segment type */
        unsigned short  xs_attr;        /* segment attributes */
        unsigned short  xs_seg;         /* segment number */
        char            xs_align;       /* log base 2 of alignment */
        char            xs_cres;        /* unused */
        long            xs_filpos;      /* file position */
        long            xs_psize;       /* physical size (in file) */
        long            xs_vsize;       /* virtual size (in core) */
        long            xs_rbase;       /* relocation base address/offset */
        unsigned short  xs_noff;        /* segment name string table offset */
        unsigned short  xs_sres;        /* unused */
        long            xs_lres;        /* unused */
};

struct xiter {                  /* x.out iteration record */
        long            xi_size;        /* source byte count */
        long            xi_rep;         /* replication count */
        long            xi_offset;      /* destination offset in segment */
};

struct xlist {                  /* xlist structure for xlist(3). */
        unsigned short  xl_type;        /* symbol type */
        unsigned short  xl_seg;         /* file segment table index */
        long            xl_value;       /* symbol value */
        char            *xl_name;       /* pointer to ascii name */
};
```

```
struct aexec {                  /* a.out header */
        unsigned short    xa_magic;     /* magic number */
        unsigned short    xa_text;      /* size of text segment */
        unsigned short    xa_data;      /* size of initialized data */
        unsigned short    xa_bss;       /* size of uninitialized data */
        unsigned short    xa_syms;      /* size of symbol table */
        unsigned short    xa_entry;     /* entry point */
        unsigned short    xa_unused;    /* not used */
        unsigned short    xa_flag;      /* relocation info stripped */
};

struct nlist {                  /* nlist structure for nlist(3). */
        char              n_name[8];    /* symbol name */
        int               n_type;       /* type flag */
        unsigned          n_value;      /* value */
};

struct bexec {                  /* b.out header */
        long              xb_magic;     /* magic number */
        long              xb_text;      /* text segment size */
        long              xb_data;      /* data segment size */
        long              xb_bss;       /* bss size */
        long              xb_syms;      /* symbol table size */
        long              xb_trsize;    /* text relocation table size */
        long              xb_drsize;    /* data relocation table size */
        long              xb_entry;     /* entry point */
};
```

See also

ld(CP), **masm**(CP), **nm**(CP), **strip**(CP), **xlist**(S)

Standards conformance

x.out is an extension of AT&T System V provided by The Santa Cruz Operation.

XENIX cross development (XNX)

Intro

introduction to XENIX cross-development commands

Description

This section describes Development System commands that are used for XENIX system cross-development.

The following is a listing of the manual pages found in this section and a brief description of each command's purpose.

Manual page	Description
ar(XNX)	maintains archives and libraries
hdr(XNX)	display selected parts of a XENIX object file
ld(XNX)	invokes the link editor
nm(XNX)	prints name list
prof(XNX)	displays profile data
ranlib(XNX)	converts archives to random libraries
size(XNX)	prints the size of an object file
strip(XNX)	removes symbols and relocation bits

ar

maintains archives and libraries

Syntax

ar *key* [*posname*] *afile names* ...

Description

ar maintains groups of files combined into a single XENIX format archive file. Its main use is to create and update library files as used by the link editor though it can be used for any similar purpose.

key is one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcln**. *afile* is the archive file. The *names* are constituent files in the archive file. These files can be any combination of XENIX format object files or COFF files. The *posname* is the name of a constituent file, and is required when certain keys are used. The meanings of the *key* characters are:

d Deletes the named files from the archive file.

r Replaces the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly appends the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece by piece.

t Prints a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Prints the named files in the archive.

m Moves the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.

x Extracts the named files. If no names are given, all files in the archive are extracted. Unless the optional character **n** is used with **x**, an extracted file's modification date will be set to the date stored in that file's archive header. In neither case does **x** alter the archive file.

v Verbose. Under the verbose option, **ar** gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with **x**, it precedes each file with a name.

c Create. Normally **ar** will create **afile** when it needs to. The create option suppresses the normal message that is produced when **afile** is created.

l Local. Normally **ar** places its temporary files in the directory */tmp*. This option causes them to be placed in the local directory.

n New. When used with the key character **x** it sets the extracted file's modification date to the current date.

When **ar** creates an archive, it always creates the header in XENIX format (see **ar**(F).

Files

/tmp/v* Temporary files

Notes

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

If the archive contains OMF binaries, **ranlib**(XNX) must be used to convert the library to the proper format. It is not necessary to run **ranlib**(XNX) if the archive contains COFF binaries.

Failure to process a library with **ranlib**, or failure to reprocess a library with **ranlib**, will cause **ld** to fail. Because generation of a library by **ar** and randomization by **ranlib** are separate, phase errors are possible. The loader **ld** warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

See also

ld(XNX), **lorder**(CP), **ar**(FP)

Standards conformance

ar is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

hdr

display selected parts of a XENIX object file

Syntax

hdr [-dhmprsSt] *file* ...

Description

hdr displays object file headers, symbol tables, and text or data relocation records in ASCII formats. This utility only works on XENIX-format binaries, such as *x.out* files or *a.out* files created with either the **-xout** or **-x2.3** options to the **cc**(CP) utility.

This utility also prints out seek positions for the various segments in the object file. *A.out*, *x.out*, and *x.out* segmented formats and archives are understood.

The symbol table format consists of six fields. In *a.out* formats the third field is missing. The first field is the symbol's index or position in the symbol table, printed in decimal. The index of the first entry is zero. The second field is the type, printed in hexadecimal. The third field is the "s_seg" field, printed in hexadecimal. The fourth field is the symbol's value in hexadecimal. The fifth field is a single character that represents the symbol's type as in **nm**(XNX), except C common is not recognized as a special case of undefined. The last field is the symbol name.

If long form relocation is present, the format consists of six fields. The first is the descriptor, printed in hexadecimal. The second is the symbol ID or index, in decimal. This field is used for external relocations as an index into the symbol table. It should reference an undefined symbol table entry. The third field is the position, or offset, within the current segment at which relocation is to take place; it is printed in hexadecimal. The fourth field is the name of the segment referenced in the relocation: text, data, bss, or EXT for external. The fifth field is the size of relocation: byte, word (2 bytes), or long. The last field will indicate, if present, that the relocation is relative.

If short form relocation is present, the format consist of three fields. The first field is the relocation command in hexadecimal. The second field contains the name of the segment referenced: text or data. The last field indicates the size of relocation: word or long.

The **hdr** options are:

-d Prints the data relocation records.

-h Prints the object file header and extended header. Each field in the header or extended header is labeled. This is the default option.

-m Prints the text and data segments. This option is similar to the **-S** option but only for text and data segments.

-p Prints the seek positions as defined by macros in the include file *<a.out.h>*.

-r Prints both text and data relocation records.

-s Prints the symbol table.

-S Prints the file segment table with a header. (Only applicable to *x.out* segmented executable files.)

-t Prints the text relocation records.

Notes

hdr is only for use with object files created for XENIX environments.

See also

a.out(F), **dump**(CP), **nm**(XNX).

ld

invokes the link editor

Syntax

ld [*options*] *file*...

Description

ld is the XENIX link editor. It creates an executable program by combining one or more object files and copying the executable result to the file *a.out*. The *file* must name an object or library file. By convention these names have the ".o" (for object) or ".a" (for archive library) extensions. If more than one name is given, the names must be separated by one or more spaces. If errors occur while linking, **ld** displays an error message; the resulting *a.out* file is unexecutable.

ld concatenates the contents of the given object files in the order given in the command line. Library files in the command line are examined only if there are unresolved external references encountered from previous object files. Library files must be in **ranlib**(XNX) format, that is, the first member must be named __.SYMDEF, which is a dictionary for the library. **ld** ignores the modification dates of the library and the __.SYMDEF entry, so if object files have been added to the library since __.SYMDEF was created, the link may result in an "invalid object module."

The library is searched iteratively to satisfy as many references as possible and only those routines that define unresolved external references are concatenated. Object and library files are processed at the point they are encountered in the argument list, so the order of files in the command line is important. In general, all object files should be given before library files. **ld** sets the entry point of the resulting program to the beginning of the first routine.

ld should be invoked using the **cc**(CP) command instead of invoking it directly. **cc** invokes **ld** as the last step of compilation, providing all the necessary C-language support routines. Invoking **ld** directly is not recommended since failure to give command line arguments in the correct order can result in errors.

There are the following options:

-A *hex-num* Creates a standalone program whose expected load address (in hexadecimal) is *hex-num*. This option sets the absolute flag in the header of the *a.out* file. Such program files can only be executed as standalone programs. Options **-A** and **-F** are mutually exclusive.

-B ***hex-num*** Sets the text selector bias to the specified hexadecimal number.

-c ***num*** Alters the default target CPU in the *x.out* header. ***num*** can be 0, 1, 2, or 3 indicating 8086, 80186, 80286 and 80386 processors, respectively. The default on 8086/80286 systems is 0. The default on 80386 systems is 3. Note that this option only alters the default; if object modules containing code for a higher numbered processor are linked, then that will take precedence over the default.

-C Causes the link editor to ignore the case of symbols.

-D ***hex-num*** Sets the data selector bias to the specified hexadecimal number.

-F ***hex-num*** Sets the size of the program stack to ***hex-num*** bytes (in hexadecimal). This option is ignored for 80386 programs which have a variable sized stack. By default 8086 programs have a variable stack located at the top of the first data segment, and 80286 programs have a fixed size 4096 byte stack. The **-F** option is incompatible with the **-A** option.

-g Includes symbolic information for **sdb**.

-i Creates separate instruction and data spaces for small model programs. When the output file is executed, the program text and data areas are allocated separate physical segments. The text portion will be read-only and shared by all users executing the file.

-La Sets advisory file locking. Advisory locking is used on files with access modes that do not require mandatory locking.

-Lm Sets mandatory file locking. Mandatory file locking is used on files that cannot be opened by more than one process at a time.

-m*name* Creates a link map file named ***name*** that includes public symbols.

-M*x* Specifies the memory model. *x* can have the following values:

s	small
m	middle
l	large
h	huge
e	mixed

-n ***num*** Truncates symbols to the length specified by ***num***.

-N ***hex-num*** Sets the page size to ***hex-num*** (which should be a multiple of 512 decimal) - the default is 400 (1024 decimal) for 80386 programs. 8086/80186/80286 programs do not normally have page-aligned *x.out* files and the default for these is 0.

-o ***name*** Sets the executable program filename to ***name*** instead of *a.out*.

-P Disables packing of segments

-r Invokes the incremental linker, */lib/ldr*, with the arguments passed to **ld** to produce a relocatable output file.

-R Ensures that the relocation table is of non-zero size. Important for 8086 compatibility.

-Rd ***hex-num*** Specify the data segment relocation offset (80386 only). ***hex-num*** is a hexadecimal number.

-Rt ***hex-num*** Specify the text segment relocation offset (80386 only) ***hex-num*** is a hexadecimal number.

-s Strips the symbol table.

-S ***num*** Sets the maximum number of segments to ***num***. If no argument is given, the default is 128.

-u ***symbol*** Designates the specified ***symbol*** as undefined.

-v ***num*** Specifies the version number. Acceptable values for ***num*** are 2, 3, or 5; 5 is the default.

Files

/bin/ld

Notes

The user must make sure that the most recent library versions have been processed with **ranlib**(XNX) before linking. If this is not done, **ld** cannot create executable programs using these libraries.

ld operates on COFF files transparently, mapping COFF symbol types to standard XENIX *x.out* symbol types when possible.

See also

ar(XNX), **masm**(CP), **cc**(CP), **ranlib**(XNX)

Standards conformance

ld is conformant with:

AT&T SVID Issue 2.

nm

prints name list

Syntax

nm [**-acgnoOprsSuv**] [**++***offset*] [*file* ...]

Description

nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. **nm** works transparently on COFF files and XENIX generated object files. **nm** translates all possible COFF symbols into standard XENIX object symbols.

If no *file* is given, the symbols in *a.out* are listed.

Each symbol name is preceded by its value in hexadecimal (blanks if undefined) and one of the letters "U" (undefined), "A" (absolute), "T" (text segment symbol), "D" (data segment symbol), "B" (bss segment symbol), "S" (segment name), "C" (common symbol), "K" (8086 common segment), or "F" (source file name). If the symbol table is in segmented format, symbol values are displayed as *segment:offset*. If the symbol is local (non-external), the type letter is in lowercase. The output is sorted alphabetically.

Options are:

-a Attempt to print the namelist of all modules in an archive library. Normally, **nm** silently ignores any library members which are not valid object modules. Using this option causes **nm** to report an error for all such modules. Note that the first member in any library which has been processed by **ranlib**(XNX) is called **__.SYMDEF** and is not a valid object module, thus the **-a** option will always produce at least one error message when used on such a library.

-c Print only C program symbols (symbols which begin with '_') as they appeared in the C program.

-g Print only global (external) symbols.

-n Sort numerically rather than alphabetically.

-o Prepend file or archive element name to each output line rather than only once.

-O Print symbol values in octal.

-p Don't sort; print in symbol-table order.

-r Sort in reverse order.

-s Sort by size of symbol and display each symbol's size instead of value. The last symbol in each text or data segment may be assigned a size of 0. This implies the **-n** option.

-S Switch the display format. If the symbol table is in segmented format, print values in non-segmented format. If not segmented, print values in segmented format. Segment offsets in 386 object modules and executable files are 32 bits rather than 16 bits.

-u Print only undefined symbols.

-v Also describe the object file and symbol table format.

Files

a.out

See Also

ar(XNX), **ar**(F), **a.out**(F)

Standards Conformance

nm is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

prof

displays profile data

Syntax

prof [**-a -l**] [*file*]

Description

prof interprets the file *mon.out* produced by the **monitor** subroutine. Under default modes, the symbol table in the named object file (*a.out* default) is read and correlated with the *mon.out* profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the **-a** option is used, all symbols are reported rather than just external symbols. If the **-l** option is used, the output is listed by symbol value rather than decreasing percentage.

To cause calls to a routine to be tallied, the **-p** option of **cc** must have been given when the file containing the routine was compiled. This option also arranges for the *mon.out* file to be produced automatically.

Files

mon.out	For profile
a.out	For namelist

Notes

Beware of quantization errors.

If you use an explicit call to **monitor**(S) you will need to make sure that the buffer size is equal to or smaller than the program size.

Warning

Profiling gives incorrect results for hybrid model 286 programs (for example those with 16 bit text pointers within modules and 32 bit text pointers between modules).

See also

monitor(S), **profil**(S), **cc**(CP)

Standards conformance

prof is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

ranlib

converts archives to random libraries.

Syntax

ranlib *archive...*

Description

ranlib converts each *archive* to a form which can be loaded more rapidly by the loader, by adding a table of contents named __.SYMDEF to the beginning of the archive. It uses **ar**(XNX) to reconstruct the archive, so sufficient temporary file space must be available in the file system containing the current directory.

Notes

ranlib should only be used on libraries containing OMF binaries created with **ar**(XNX). If **ranlib** is used on a library containing COFF binaries, it will generate an error message.

Failure to process a library with **ranlib**, or failure to reprocess a library with **ranlib**, will cause **ld** to fail. Because generation of a library by **ar** and randomization by **ranlib** are separate, phase errors are possible. The loader **ld** warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

See also

ld(XNX), **ar**(XNX), **copy**(C), **settime**(ADM)

Standards conformance

ranlib is an extension of AT&T System V provided by the Santa Cruz Operation.

size

prints the size of an object file

Syntax

size [*object* ...]

Description

size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in decimal and hexadecimal, of each object-file argument. If no file is specified, *a.out* is used.

See also

a.out(F)

Standards conformance

size is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

strip

removes symbols and relocation bits

Syntax

strip [**-MNSdehrstx**] *file* ...

Description

strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and link editor. This is useful for saving space after a program has been debugged.

If *name* is an archive file, **strip** will remove the local symbols from any *a.out* format files it finds in the archive. Certain libraries, such as those residing in */lib*, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

There are several options for use with **strip**:

-M Strip all memory image segments.

-N Strip all non-memory image segments.

-S Strip the segment table only.

-h Strip header and extended header.

-e Strip extended header.

-d Strip data and data relocation.

-t Strip text and text relocation.

-r Strip all relocation except x.out's "short form."

-x Strip all relocation.

-s trip symbol table.

The effect of **strip** is the same as use of the **-s** option of **ld**.

Files

/tmp/stm* Temporary file

See Also

ld(XNX)

Standards Conformance

strip is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

DOS cross development (DOS)

Routines

DOS routines and man pages listed

Description

All DOS manual pages are printed in the *Programmer's Reference Manual* and are not available through the **man**(C) command. For information on DOS commands, see the appropriate manual.

The following table lists the routines that can be used in DOS applications. The comments in the "Note" column mean the following:

same
: routine has the same syntax, behavior, and return values in System V libraries and in DOS or OS/2 libraries

DOS only
: routine is supported only in the DOS libraries. See the DOS documentation for information about syntax, behavior, and return values.

OS/2 only
: routine is supported only in the OS/2 libraries. See the OS/2 documentation for information about syntax, behavior, and return values.

DOS-OS/2 only
: routine is supported only in the DOS and OS/2 libraries. See the DOS and OS/2 documentation for information about syntax, behavior, and return values.

differs under DOS-OS/2
: routine exists in both System V/386 libraries and in DOS or OS/2 libraries but has different syntax, behavior, and return values. See the DOS and OS/2 documentation and the appropriate section (S) manual page for details.

DOS routines/macros

Routine/macros(*)	Manual page	Note
abort(DOS)	abort(DOS)	differs under DOS-OS/2
abs(DOS)	abs(S)	same
access(DOS)	access(DOS)	differs under DOS-OS/2
acos(DOS)	trig(S)	same
acosl	acosl	DOS-OS/2 only
alloca(DOS)	alloca(DOS)	DOS-OS/2 only
_arc(DOS)	_arc(DOS)	DOS only
_arc_w(DOS)	_arc_w(DOS)	DOS only
_arc_wxy(DOS)	_arc_wxy(DOS)	DOS only
asctime(DOS)	ctime(S)	same
asin(DOS)	asin(S)	same
asinl(DOS)	asinl(DOS)	DOS-OS/2 only
assert(DOS)(*)	assert(S)	same
atan(DOS)	trig(S)	same
atanl(DOS)	atanl(DOS)	DOS-OS/2 only
atan2(DOS)	trig(S)	same
atan2l(DOS)	atan2l(DOS)	DOS-OS/2 only
atexit(DOS)	atexit(S)	same
atof(DOS)	atof(S)	same
atoi(DOS)	atof(S)	same
atol(DOS)	atof(S)	same
_atold(DOS)	_atold(DOS)	DOS-OS/2 only
_bcalloc(DOS)	_bcalloc(DOS)	DOS-OS/2 only
bdos(DOS)	bdos(DOS)	DOS only
_beginthread(DOS)	_beginthread(DOS)	OS/2 only
_bexpand(DOS)	_bexpand(DOS)	DOS-OS/2 only
_bfree(DOS)	_bfree(DOS)	DOS-OS/2 only
_bfreeseg(DOS)	_bfreeseg(DOS)	DOS-OS/2 only
_bheapadd(DOS)	_bheapadd(DOS)	DOS-OS/2 only
_bheapchk(DOS)	_bheapchk(DOS)	DOS-OS/2 only
_bheapmin(DOS)	_bheapmin(DOS)	DOS-OS/2 only
_bheapseg(DOS)	_bheapseg(DOS)	DOS-OS/2 only
_bheapset(DOS)	_bheapset(DOS)	DOS-OS/2 only
_bheapwalk(DOS)	_bheapwalk(DOS)	DOS-OS/2 only
_bios_disk(DOS)	_bios_disk(DOS)	DOS only
_bios_equiplist(DOS)	_bios_equiplist(DOS)	DOS only
_bios_keybrd(DOS)	_bios_keybrd(DOS)	DOS only
_bios_memsize(DOS)	_bios_memsize(DOS)	DOS only
_bios_printer(DOS)	_bios_printer(DOS)	DOS only
_bios_serialcom(DOS)	_bios_serialcom(DOS)	DOS only
_bios_timeofday(DOS)	_bios_timeofday(DOS)	DOS only
_bmalloc(DOS)	_bmalloc(DOS)	DOS-OS/2 only
_bmsize(DOS)	_bmsize(DOS)	DOS-OS/2 only
_brealloc(DOS)	_brealloc(DOS)	DOS-OS/2 only
bsearch(DOS)	bsearch(S)	same
cabs(DOS)	cabs(DOS)	DOS-OS/2 only
cabsl(DOS)	cabsl(DOS)	DOS-OS/2 only
calloc(DOS)	calloc(S)	same
calloc(DOS)	calloc(S)	same

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
ceil(DOS)	floor(S)	same
ceill(DOS)	ceill(DOS)	DOS-OS/2 only
_cexit(DOS)	_cexit(DOS)	DOS-OS/2 only
_c_exit(DOS)	_c_exit(DOS)	DOS-OS/2 only
cgets(DOS)	cgets(DOS)	DOS-OS/2 only
_chain_intr(DOS)	_chain_intr(DOS)	DOS only
chdir(DOS)	chdir(DOS)	differs under DOS-OS/2 only
_chdrive(DOS)	_chdrive(DOS)	DOS-OS/2 only
chmod(DOS)	chmod(DOS)	differs under DOS-OS/2
chsize(DOS)	chsize(S)	same
_clear87(DOS)	_clear87(DOS)	DOS-OS/2 only
clearerr(DOS)	ferror(S)	same
_clearscreen(DOS)	_clearscreen(DOS)	DOS-OS/2 only
clock(DOS)	clock(S)	same
close(DOS)	close(S)	same
_control87(DOS)	_control87(DOS)	DOS-OS/2 only
cos(DOS)	trig(S)	same
cosh(DOS)	sinh(S)	same
coshl(DOS)	coshl(DOS)	DOS-OS/2 only
cosl(DOS)	cosl(DOS)	DOS-OS/2 only
cprintf(DOS)	cprintf(DOS)	DOS-OS/2 only
cputs(DOS)	cputs(DOS)	DOS-OS/2 only
creat(DOS)	creat(DOS)	differs under DOS-OS/2
cscanf(DOS)	cscanf(DOS)	DOS-OS/2 only
ctime(DOS)	ctime(S)	same
cwait(DOS)	cwait(DOS)	OS/2 only
dieeetomsbin(DOS)	dieeetomsbin(DOS)	DOS-OS/2 only
difftime(DOS)	difftime(S)	same
div(DOS)	div(S)	same
_disable(DOS)	_disable(DOS)	DOS only
_displaycursor(DOS)	_displaycursor(DOS)	DOS-OS/2 only
dmsbintoieee(DOS)	dieeetomsbin(DOS)	DOS-OS/2 only
_dos_allocmem(DOS)	_dos_allocmem(DOS)	DOS only
_dos_close(DOS)	_dos_close(DOS)	DOS only
_dos_creat(DOS)	_dos_creat(DOS)	DOS only
_dos_creatnew(DOS)	_dos_creatnew(DOS)	DOS only
dosexterr(DOS)	dosexterr(DOS)	DOS only
_dos_findfirst(DOS)	_dos_findfirst(DOS)	DOS only
_dos_findnext(DOS)	_dos_findnext(DOS)	DOS only
_dos_freemem(DOS)	_dos_freemem(DOS)	DOS only
_dos_getdate(DOS)	_dos_getdate(DOS)	DOS only
_dos_getdiskfree(DOS)	_dos_getdiskfree(DOS)	DOS only
_dos_getdrive(DOS)	_dos_getdrive(DOS)	DOS only
_dos_getfileattr(DOS)	_dos_getfileattr(DOS)	DOS only
_dos_getftime(DOS)	_dos_getftime(DOS)	DOS only
_dos_gettime(DOS)	_dos_gettime(DOS)	DOS only
_dos_getvect(DOS)	_dos_getvect(DOS)	DOS only
_dos_keep(DOS)	_dos_keep(DOS)	DOS only

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
_dos_open(DOS)	_dos_open(DOS)	DOS only
_dos_read(DOS)	_dos_read(DOS)	DOS only
_dos_setblock(DOS)	_dos_setblock(DOS)	DOS only
_dos_setdate(DOS)	_dos_setdate(DOS)	DOS only
_dos_setdrive(DOS)	_dos_setdrive(DOS)	DOS only
_dos_setfileattr(DOS)	_dos_setfileattr(DOS)	DOS only
_dos_setftime(DOS)	_dos_setftime(DOS)	DOS only
_dos_settime(DOS)	_dos_settime(DOS)	DOS only
_dos_setvect(DOS)	_dos_setvect(DOS)	DOS only
_dos_write(DOS)	_dos_write(DOS)	DOS only
dup(DOS)	dup(S)	same
dup2(DOS)	dup(S)	same
ecvt(DOS)	ecvt(S)	same
_ellipse(DOS)	_ellipse(DOS)	DOS only
_ellipse_w(DOS)	_ellipse_w(DOS)	DOS only
_ellipse_wxy(DOS)	_ellipse_wxy(DOS)	DOS only
_enable(DOS)	_enable(DOS)	DOS only
_endthread(DOS)	_endthread(DOS)	OS/2 only
eof(DOS)	eof(DOS)	DOS-OS/2 only
execl(DOS)	exec(DOS)	differs under DOS-OS/2
execle(DOS)	exec(DOS)	differs under DOS-OS/2
execlp(DOS)	exec(DOS)	differs under DOS-OS/2
execlpe(DOS)	exec(DOS)	DOS-OS/2 only
execv(DOS)	exec(DOS)	differs under DOS-OS/2
execve(DOS)	exec(DOS)	differs under DOS-OS/2
execvp(DOS)	exec(DOS)	differs under DOS-OS/2
execvpe(DOS)	exec(DOS)	DOS-OS/2 only
exit(DOS)	exit(DOS)	differs under DOS-OS/2
_exit(DOS)	_exit(DOS)	DOS-OS/2 only
exp(DOS)	exp(S)	same
_expand(DOS)	_expand(DOS)	DOS-OS/2 only
expl(DOS)	expl(DOS)	DOS-OS/2 only
fabs(DOS)	floor(S)	same
fabsl(DOS)	fabsl(DOS)	DOS-OS/2 only
_fcalloc(DOS)	_fcalloc(DOS)	DOS-OS/2 only
fclose(DOS)	fclose(DOS)	differs under DOS-OS/2
fcloseall(DOS)	fclose(DOS)	DOS-OS/2 only
fcvt(DOS)	ecvt(S)	same
fdopen(DOS)	fopen(S)	same
feof(DOS)	ferror(S)	same
ferror(DOS)	ferror(S)	same
_fexpand(DOS)	_fexpand(DOS)	DOS-OS/2 only
fflush(DOS)	fflush(S)	same
_ffree(DOS)	free(DOS)	DOS-OS/2 only
fgetc(DOS)	fgetc(DOS)	differs under DOS-OS/2
fgetchar(DOS)	fgetc(DOS)	DOS-OS/2 only
fgetpos(DOS)	fgetpos(S)	same
fgets(DOS)	fgets(S)	same

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
_fheapchk(DOS)	_fheapchk(DOS)	DOS-OS/2 only
_fheapwalk(DOS)	_fheapwalk(DOS)	DOS-OS/2 only
_fheapmin(DOS)	_fheapmin(DOS)	DOS-OS/2 only
_fheapset(DOS)	_fheapset(DOS)	DOS-OS/2 only
fieeetomsbin(DOS)	fieeetomsbin(DOS)	DOS-OS/2 only
filelength(DOS)	filelength(DOS)	DOS-OS/2 only
fileno(DOS)	ferror(S)	same
_floodfill(DOS)	_floodfill(DOS)	DOS only
_floodfill_w(DOS)	_floodfill_w(DOS)	DOS only
floor(DOS)	floor(S)	same
floorl(DOS)	floorl(DOS)	DOS-OS/2 only
flushall(DOS)	flushall(DOS)	differs under DOS-OS/2
_fmalloc(DOS)	malloc(DOS)	DOS-OS/2 only
_fmemccpy(DOS)	memccpy(DOS)	DOS-OS/2 only
_fmemchr(DOS)	memchr(DOS)	DOS-OS/2 only
_fmemcmp(DOS)	memcmp(DOS)	DOS-OS/2 only
_fmemcpy(DOS)	memcpy(DOS)	DOS-OS/2 only
_fmemicmp(DOS)	memicmp(DOS)	DOS-OS/2 only
_fmemmove(DOS)	memmove(DOS)	DOS-OS/2 only
_fmemset(DOS)	memset(DOS)	DOS-OS/2 only
fmod(DOS)	floor(S)	same
fmodl(DOS)	fmodl(DOS)	DOS-OS/2 only
fmsbintoieee(DOS)	fieeetomsbin(DOS)	DOS-OS/2 only
_fmsize(DOS)	movedata(DOS)	DOS-OS/2 only
fopen(DOS)	fopen(DOS)	differs under DOS-OS/2
FP_OFF(DOS)(*)	fopen(DOS)	DOS-OS/2 only
_fpreset(DOS)	_fpreset(DOS)	DOS-OS/2 only
fprintf(DOS)	printf(S)	same
FP_SEG(DOS)(*)	fopen(DOS)	DOS-OS/2 only
fputc(DOS)	fputc(DOS)	differs under DOS-OS/2
fputchar(DOS)	fputc(DOS)	DOS-OS/2 only
fputs(DOS)	fputs(S)	same
fread(DOS)	fread(DOS)	differs under DOS-OS/2
_frealloc(DOS)	_frealloc(DOS)	DOS-OS/2 only
free(DOS)	free(S)	same
_freect(DOS)	_freect(DOS)	DOS-OS/2 only
freopen(DOS)	freopen(DOS)	differs under DOS-OS/2
frexp(DOS)	frexp(S)	same
frexpl(DOS)	frexpl(DOS)	DOS-OS/2 only
fscanf(DOS)	fscanf(S)	same
fseek(DOS)	fseek(DOS)	differs under DOS-OS/2
fsetpos(DOS)	fsetpos(S)	same
_fsopen(DOS)	_fsopen(DOS)	DOS-OS/2 only
fstat(DOS)	fstat(DOS)	differs under DOS-OS/2
_fstrcat(DOS)	strcat(DOS)	DOS-OS/2 only
_fstrchr(DOS)	strchr(DOS)	DOS-OS/2 only
_fstrcmp(DOS)	strcmp(DOS)	DOS-OS/2 only
_fstrcpy(DOS)	strcpy(DOS)	DOS-OS/2 only

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
_fstrcspn(DOS)	strcspn(DOS)	DOS-OS/2 only
_fstrdup(DOS)	strdup(DOS)	DOS-OS/2 only
_fstricmp(DOS)	stricmp(DOS)	DOS-OS/2 only
_fstrlen(DOS)	strlen(DOS)	DOS-OS/2 only
_fstrlwr(DOS)	strlwr(DOS)	DOS-OS/2 only
_fstrncat(DOS)	strncat(DOS)	DOS-OS/2 only
_fstrncmp(DOS)	strncmp(DOS)	DOS-OS/2 only
_fstrncpy(DOS)	strncpy(DOS)	DOS-OS/2 only
_fstrnicmp(DOS)	strnicmp(DOS)	DOS-OS/2 only
_fstrnset(DOS)	strnset(DOS)	DOS-OS/2 only
_fstrpbrk(DOS)	strpbrk(DOS)	DOS-OS/2 only
_fstrrchr(DOS)	strrchr(DOS)	DOS-OS/2 only
_fstrrev(DOS)	strrev(DOS)	DOS-OS/2 only
_fstrset(DOS)	strset(DOS)	DOS-OS/2 only
_fstrspn(DOS)	strspn(DOS)	DOS-OS/2 only
_fstrstr(DOS)	strstr(DOS)	DOS-OS/2 only
_fstrtok(DOS)	strtok(DOS)	DOS-OS/2 only
_fstrupr(DOS)	strupr(DOS)	DOS-OS/2 only
ftell(DOS)	ftell(DOS)	differs under DOS-OS/2
ftime(DOS)	ftime(DOS)	differs under DOS-OS/2
_fullpath(DOS)	_fullpath(DOS)	DOS-OS/2 only
fwrite(DOS)	fwrite(DOS)	differs under DOS-OS/2
gcvt(DOS)	ecvt(S)	same
_getactivepage(DOS)	_getactivepage(DOS)	DOS-OS/2 only
_getarcinfo(DOS)	_getarcinfo(DOS)	DOS only
_getbkcolor(DOS)	_getbkcolor(DOS)	DOS-OS/2 only
getc(DOS)	getc(S)	same
getch(DOS)	getch(DOS)	DOS-OS/2 only
getchar(DOS)	getc(S)	same
getche(DOS)	getch(DOS)	DOS-OS/2 only
_getcolor(DOS)	_getcolor(DOS)	DOS only
_getcurrentposition(DOS)	_getcurrentposition(DOS)	DOS only
_getcurrentposition_w(DOS)	_getcurrentposition_w(DOS)	DOS-OS/2 only
getcwd(DOS)	getcwd(S)	same
_getdcwd(DOS)	_getdcwd(DOS)	DOS-OS/2 only
_getdrive(DOS)	_getdrive(DOS)	DOS-OS/2 only
getenv(DOS)	getenv(S)	same
_getfillmask(DOS)	_getfillmask(DOS)	DOS only
_getfontinfo(DOS)	_getfontinfo(DOS)	DOS only
_getgtextextent(DOS)	_getgtextextent(DOS)	DOS only
_getgtextvetor(DOS)	_getgtextvetor(DOS)	DOS only
_getimage(DOS)	_getimage(DOS)	DOS only
_getimage_w(DOS)	_getimage_w(DOS)	DOS only
_getimage_wxy(DOS)	_getimage_wxy(DOS)	DOS only
_getlinestyle(DOS)	_getlinestyle(DOS)	DOS only
_getphyscoord(DOS)	_getphyscoord(DOS)	DOS only
getpid(DOS)	getpid(DOS)	differs under DOS-OS/2
_getpixel(DOS)	_getpixel(DOS)	DOS only

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
_getpixel_w(DOS)	_getpixel_w(DOS)	DOS only
gets(DOS)	gets(S)	same
_gettextcolor(DOS)	_gettextcolor(DOS)	DOS-OS/2 only
_gettextcursor(DOS)	_gettextcursor(DOS)	DOS-OS/2 only
_gettextposition(DOS)	_gettextposition(DOS)	DOS-OS/2 only
_gettextwindow(DOS)	_gettextwindow(DOS)	DOS-OS/2 only
_getvideoconfig(DOS)	_getvideoconfig(DOS)	DOS-OS/2 only
_getviewcoord(DOS)	_getviewcoord(DOS)	DOS-OS/2 only
_getviewcoord_w(DOS)	_getviewcoord_w(DOS)	DOS-OS/2 only
_getviewcoord_wxy(DOS)	_getviewcoord_wxy(DOS)	DOS-OS/2 only
_getvisualpage(DOS)	_getvisualpage(DOS)	DOS-OS/2 only
getw(DOS)	getw(S)	same
_getwindowcoord(DOS)	_getwindowcoord(DOS)	DOS-OS/2 only
_getwritemode(DOS)	_getwritemode(DOS)	DOS-OS/2 only
gmtime(DOS)	gmtime(S)	same
_grstatus(DOS)	_grstatus(DOS)	DOS-OS/2 only
halloc(DOS)	halloc(DOS)	DOS-OS/2 only
_harderr(DOS)	_harderr(DOS)	DOS only
_hardresume(DOS)	_hardresume(DOS)	DOS only
_hardretn(DOS)	_hardretn(DOS)	DOS only
_heapadd(DOS)	_heapadd(DOS)	DOS-OS/2 only
_heapchk(DOS)	_heapchk(DOS)	DOS-OS/2 only
_heapmin(DOS)	_heapmin(DOS)	DOS-OS/2 only
_heapset(DOS)	_heapset(DOS)	DOS-OS/2 only
_heapwalk(DOS)	_heapwalk(DOS)	DOS-OS/2 only
hfree(DOS)	hfree(DOS)	DOS-OS/2 only
hypot(DOS)	hypot(S)	same
hypotl(DOS)	hypotl(DOS)	DOS-OS/2 only
_imagesize(DOS)	_imagesize(DOS)	DOS only
_imagesize_w(DOS)	_imagesize_w(DOS)	DOS only
_imagesize_wxy(DOS)	_imagesize_wxy(DOS)	DOS only
inp(DOS)	inp(DOS)	DOS-OS/2 only
inpw(DOS)	inpw(DOS)	DOS-OS/2 only
int86(DOS)	int86(DOS)	DOS only
int86x(DOS)	int86x(DOS)	DOS only
intdos(DOS)	intdos(DOS)	DOS only
intdosx(DOS)	intdosx(DOS)	DOS only
isalnum(DOS)	ctype(S)	same
isalpha(DOS)	ctype(S)	same
isascii(DOS)	ctype(S)	same
isatty(DOS)	isatty(DOS)	differs under DOS-OS/2
iscntrl(DOS)	ctype(S)	same
isdigit(DOS)	ctype(S)	same
isgraph(DOS)	ctype(S)	same
islower(DOS)	ctype(S)	same
isprint(DOS)	ctype(S)	same
ispunct(DOS)	ctype(S)	same
isspace(DOS)	ctype(S)	same

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
isupper(DOS)	ctype(S)	same
isxdigit(DOS)	ctype(S)	same
itoa(DOS)	itoa(DOS)	DOS-OS/2 only
j0(DOS)	bessel(S)	same
_j0l(DOS)	_j0l(DOS)	DOS-OS/2 only
j1(DOS)	bessel(S)	same
_j1l(DOS)	_j1l(DOS	DOS-OS/2 only
jn(DOS)	bessel(S)	same
kbhit(DOS)	kbhit(DOS)	DOS-OS/2 only
labs(DOS)	labs(DOS)	DOS-OS/2 only
ldexp(DOS)	frexp(S)	same
ldexpl(DOS)	ldexpl(DOS)	DOS-OS/2 only
ldiv(DOS)	ldiv(S)	same
lfind(DOS)	lfind(DOS)	DOS-OS/2 only
_lineto(DOS)	_lineto(DOS)	DOS only
_lineto_w(DOS)	_lineto_w(DOS)	DOS only
localeconv(DOS)	localeconv(DOS)	DOS-OS/2 only
localtime(DOS)	ctime(S)	same
locking(DOS)	locking(DOS)	differs under DOS-OS/2
log(DOS)	exp(S)	same
logl(DOS)	logl(DOS)	DOS-OS/2 only
log10(DOS)	exp(S)	same
log10l(DOS)	log10l(DOS)	DOS-OS/2 only
longjmp(DOS)	setjmp(S)	same
_lrotl(DOS)	_lrotl(DOS)	DOS-OS/2 only
_lrotr(DOS)	_lrotr(DOS)	DOS-OS/2 only
lsearch(DOS)	lfind(DOS)	DOS-OS/2 only
lseek(DOS)	lseek(DOS)	differs under DOS-OS/2
ltoa(DOS)	ltoa(DOS)	DOS-OS/2 only
_makepath(DOS)	_makepath(DOS)	DOS-OS/2 only
malloc(DOS)	malloc(DOS)	differs under DOS-OS/2
matherr(DOS)	matherr(S)	same
_matherr(DOS)	_matherr(DOS)	DOS-OS/2
max(DOS)(*)	max(DOS)	DOS-OS/2
_memavl(DOS)	_memavl(DOS)	DOS-OS/2 only
memccpy(DOS)	memory(S)	same
memchr(DOS)	memory(S)	same
memcmp(DOS)	memory(S)	same
memcpy(DOS)	memory(S)	same
memicmp(DOS)	memicmp(DOS)	DOS-OS/2 only
_memmax(DOS)	_memmax(DOS)	DOS-OS/2 only
memmove(DOS)	memmove(S)	same
memset(DOS)	memory(S)	same
min(DOS)(*)	min(DOS)	DOS-OS/2 only
mkdir(DOS)	mkdir(DOS)	differs under DOS-OS/2
mktemp(DOS)	mktemp(S)	same
mktime(DOS)	mktime(S)	same
modf(DOS)	frexp(S)	same

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
modfl(DOS)	modfl(DOS)	DOS-OS/2 only
movedata(DOS)	movedata(DOS)	DOS-OS/2 only
_moveto(DOS)	_moveto(DOS)	DOS only
_moveto_w(DOS)	_moveto_w(DOS)	DOS only
_msize(DOS)	_msize(DOS)	DOS-OS/2 only
_ncalloc(DOS)	_ncalloc(DOS)	DOS-OS/2 only
_nexpand(DOS)	_nexpand(DOS)	DOS-OS/2 only
_nfree(DOS)	free(DOS)	DOS-OS/2 only
_nheapchk(DOS)	_nheapchk(DOS)	DOS-OS/2 only
_nheapmin(DOS)	_nheapmin(DOS)	DOS-OS/2 only
_nheapset(DOS)	_nheapset(DOS)	DOS-OS/2 only
_nheapwalk(DOS)	_nheapwalk(DOS)	DOS-OS/2 only
_nmalloc(DOS)	malloc(DOS)	DOS-OS/2 only
_nmsize(DOS)	_msize(DOS)	DOS-OS/2 only
_nrealloc(DOS)	_nrealloc(DOS)	DOS-OS/2 only
_nstrdup(DOS)	_nstrdup(DOS)	DOS-OS/2 only
onexit(DOS)	onexit(DOS)	DOS-OS/2 only
open(DOS)	open(DOS)	differs under DOS-OS/2
_outgtext(DOS)	_outgtext(DOS)	DOS-OS/2 only
_outmem(DOS)	_outmem(DOS)	DOS-OS/2 only
outp(DOS)	outp(DOS)	DOS-OS/2 only
outpw(DOS)	outp(DOS)	DOS-OS/2 only
_outtext(DOS)	_outtext(DOS)	DOS-OS/2 only
_pclose(DOS)	_pclose(DOS)	DOS-OS/2 only
perror(DOS)	perror(S)	same
_pg_analyzechart(DOS)	_pg_analyzechart(DOS)	DOS only
_pg_analyzechartms(DOS)	_pg_analyzechartms(DOS)	DOS only
_pg_analyzepie(DOS)	_pg_analyzepie(DOS)	DOS only
_pg_analyzescatter(DOS)	_pg_analyzescatter(DOS)	DOS only
_pg_analyzescatterms(DOS)	_pg_analyzescatterms(DOS)	DOS only
_pg_chart(DOS)	_pg_chart(DOS)	DOS only
_pg_chart(DOS)	_pg_chart(DOS)	DOS only
_pg_chartpie(DOS)	_pg_chartpie(DOS)	DOS only
_pg_chartscatter(DOS)	_pg_chartscatter(DOS)	DOS only
_pg_chartscatterms(DOS)	_pg_chartscatterms(DOS)	DOS only
_pg_defaultchart(DOS)	_pg_defaultchart(DOS)	DOS only
_pg_getchardef(DOS)	_pg_getchardef(DOS)	DOS only
_pg_getpalette(DOS)	_pg_getpalette(DOS)	DOS only
_pg_getstyleset(DOS)	_pg_getstyleset(DOS)	DOS only
_pg_hlabelchart(DOS)	_pg_hlabelchart(DOS)	DOS only
_pg_initchart(DOS)	_pg_initchart(DOS)	DOS only
_pg_resetpalette(DOS)	_pg_resetpalette(DOS)	DOS only
_pg_resetstyleset(DOS)	_pg_resetstyleset(DOS)	DOS only
_pg_setchardef(DOS)	_pg_setchardef(DOS)	DOS only
_pg_setpalette(DOS)	_pg_setpalette(DOS)	DOS only
_pg_setstyleset(DOS)	_pg_setstyleset(DOS)	DOS only
_pg_vlabelchart(DOS)	_pg_vlabelchart(DOS)	DOS only
_pie(DOS)	_pie(DOS)	DOS only

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
_pie_w(DOS)	_pie_w(DOS)	DOS only
_pie_wxy(DOS)	_pie_wxy(DOS)	DOS only
_pipe(DOS)	_pipe(DOS)	DOS-OS/2 only
_polygon(DOS)	_polygon(DOS)	DOS only
_polygon_w(DOS)	_polygon_w(DOS)	DOS only
_polygon_wxy(DOS)	_polygon_wxy(DOS)	DOS only
_popen(OS/2)	_popen(OS/2)	OS/2 only
pow(DOS)	pow(S)	same
powl(DOS)	powl(DOS)	DOS-OS/2 only
printf(DOS)	printf(S)	same
putc(DOS)	putc(S)	same
putch(DOS)	putch(DOS)	DOS-OS/2 only
putchar(DOS)	putc(S)	same
putenv(DOS)	putenv(S)	same
_putimage(DOS)	_putimage(DOS)	DOS only
_putimage_w(DOS)	_putimage_w(DOS)	DOS only
puts(DOS)	puts(S)	same
putw(DOS)	putc(S)	same
qsort(DOS)	qsort(S)	same
raise(DOS)	raise(DOS)	DOS-OS/2 only
rand(DOS)	rand(S)	same
read(DOS)	read(DOS)	differs under DOS-OS/2
realloc(DOS)	malloc(S)	same
_rectangle(DOS)	_rectangle(DOS)	DOS only
_rectangle_w(DOS)	_rectangle_w(DOS)	DOS only
_rectangle_wxy(DOS)	_rectangle_wxy(DOS)	DOS only
_registerfonts(DOS)	_registerfonts(DOS)	DOS only
_remapallpalette(DOS)	_remapallpalette(DOS)	DOS only
_remappalette(DOS)	_remappalette(DOS)	DOS only
remove(DOS)	remove(S)	same
rename(DOS)	rename(DOS)	DOS-OS/2 only
rewind(DOS)	rewind(S)	same
rmdir(DOS)	rmdir(DOS)	DOS-OS/2 only
rmtmp(DOS)	rmtmp(DOS)	DOS-OS/2 only
_rotl(DOS)	_rotl(DOS)	DOS-OS/2 only
_rotr(DOS)	_rotr(DOS)	DOS-OS/2 only
scanf(DOS)	scanf(S)	same
_scrolltextwindow(DOS)	_scrolltextwindow(DOS)	DOS-OS/2 only
_searchenv(DOS)	_searchenv(DOS)	DOS-OS/2 only
segread(DOS)	segread(DOS)	DOS-OS/2 only
_selectpalette(DOS)	_selectpalette(DOS)	DOS only
_setactivepage(DOS)	_setactivepage(DOS)	DOS-OS/2 only
_setbkcolor(DOS)	_setbkcolor(DOS)	DOS-OS/2 only
setbuf(DOS)	setbuf(S)	same
setjmp(DOS)	setjmp(S)	same
setmode(DOS)	setmode(DOS)	DOS-OS/2 only
setvbuf(DOS)	setvbuf(DOS)	DOS-OS/2 only
_setcliprgn(DOS)	_setcliprgn(DOS)	DOS only

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
_setcolor(DOS)	_setcolor(DOS)	DOS only
_setfillmask(DOS)	_setfillmask(DOS)	DOS only
_setfont(DOS)	_setfont(DOS)	DOS only
_setgtextvector(DOS)	_setgtextvector(DOS)	DOS only
_setlinestyle(DOS)	_setlinestyle(DOS)	DOS only
setlocale(DOS)	setlocale(DOS)	DOS only
_setpixel(DOS)	_setpixel(DOS)	DOS only
_setpixel_w(DOS)	_setpixel_w(DOS)	DOS only
_settextcolor(DOS)	_settextcolor(DOS)	DOS-OS/2 only
_settextcursor(DOS)	_settextcursor(DOS)	DOS-OS/2 only
_settextposition(DOS)	_settextposition(DOS)	DOS-OS/2 only
_settextrows(DOS)	_settextrows(DOS)	DOS-OS/2 only
_settextwindow(DOS)	_settextwindow(DOS)	DOS-OS/2 only
_setvideomode(DOS)	_setvideomode(DOS)	DOS-OS/2 only
_setvideomoderows(DOS)	_setvideomoderows(DOS)	DOS-OS/2 only
_setvieworg(DOS)	_setvieworg(DOS)	DOS only
_setviewport(DOS)	_setviewport(DOS)	DOS only
_setvisualpage(DOS)	_setvisualpage(DOS)	DOS only
_setwindow(DOS)	_setwindow(DOS)	DOS-OS/2 only
_setwritemode(DOS)	_setwritemode(DOS)	DOS only
signal(DOS)	signal(DOS)	differs under DOS-OS/2
sin(DOS)	trig(S)	same
sinh(DOS)	sinh(S)	same
sinhl(DOS)	sinhl(DOS)	DOS-OS/2 only
sinl(DOS)	sinl(DOS)	DOS-OS/2 only
sopen(DOS)	sopen(DOS)	DOS-OS/2 only
spawnl(DOS)	spawn(DOS)	DOS-OS/2 only
spawnle(DOS)	spawn(DOS)	DOS-OS/2 only
spawnlp(DOS)	spawn(DOS)	DOS-OS/2 only
spawnlpe(DOS)	spawn(DOS)	DOS-OS/2 only
spawnv(DOS)	spawn(DOS)	DOS-OS/2 only
spawnve(DOS)	spawn(DOS)	DOS-OS/2 only
spawnvp(DOS)	spawn(DOS)	DOS-OS/2 only
spawnvpe(DOS)	spawn(DOS)	DOS-OS/2 only
_splitpath(DOS)	_splitpath(DOS)	DOS-OS/2 only
sprintf(DOS)	printf(S)	same
sqrt(DOS)	exp(S)	same
srand(DOS)	rand(S)	same
sscanf(DOS)	scanf(S)	same
stackavail(DOS)	stackavail(DOS)	DOS-OS/2 only
stat(DOS)	stat(DOS)	differs under DOS-OS/2
_status87(DOS)	_status87(DOS)	DOS-OS/2 only
strcat(DOS)	strcat(S)	same
strchr(DOS)	strcat(S)	same
strcmp(DOS)	strcat(DOS)	DOS-OS/2 only
strcoll(DOS)	strcoll(DOS)	DOS-OS/2 only
strcpy(DOS)	strcat(S)	same
strcspn(DOS)	strcat(DOS)	differs under DOS-OS/2

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
_strdate(DOS)	_strdate(DOS)	DOS-OS/2 only
strdup(DOS)	strcat(S)	same
strerror(DOS)	strerror(S)	same
_strerror(DOS)	_strerror(DOS)	DOS-OS/2 only
strftime(DOS)	strftime(DOS)	DOS-OS/2 only
stricmp(DOS)	strcat(DOS)	DOS-OS/2 only
strlen(DOS)	string(S)	same
strlwr(DOS)	strlwr(DOS)	DOS-OS/2 only
strncat(DOS)	strncat(DOS)	differs under DOS-OS/2
strncmp(DOS)	strncat(DOS)	differs under DOS-OS/2
strncpy(DOS)	strncat(DOS)	differs under DOS-OS/2
strnicmp(DOS)	strncat(DOS)	DOS-OS/2 only
strnset(DOS)	strncat(DOS)	DOS-OS/2 only
strpbrk(DOS)	string(S)	same
strrchr(DOS)	string(S)	same
strrev(DOS)	strrev(DOS)	DOS-OS/2 only
strset(DOS)	strset(DOS)	DOS-OS/2 only
strspn(DOS)	string(S)	same
strstr(DOS)	strstr(DOS)	DOS-OS/2 only
_strtime(DOS)	_strtime(DOS)	DOS-OS/2 only
strtod(DOS)	strtod(S)	same
strtok(DOS)	string(S)	same
strtol(DOS)	strtol(S)	same
_strtold(DOS)	_strtold(S)	same
strtoul(DOS)	strtoul(DOS)	DOS-OS/2 only
strupr(DOS)	strupr(DOS)	DOS-OS/2 only
strxfrm(DOS)	strxfrm(DOS)	DOS-OS/2 only
swab(DOS)	swab(S)	same
system(DOS)	system(DOS)	differs under DOS-OS/2
tan(DOS)	trig(S)	same
tanh(DOS)	sinh(S)	same
tanl(DOS)	tanl(DOS)	DOS-OS/2 only
tanhl(DOS)	tanhl(DOS)	DOS-OS/2 only
tell(DOS)	tell(DOS)	differs under DOS-OS/2
tempnam(DOS)	tempnam(S)	same
time(DOS)	time(S)	same
tmpfile(DOS)	tmpfile(S)	same
tmpnam(DOS)	tempnam(S)	same
toascii(DOS)	ctype(S)	same
tolower(DOS)	ctype(S)	same
_tolower(DOS)	ctype(S)	same
toupper(DOS)	ctype(S)	same
_toupper(DOS)	ctype(S)	same
tzset(DOS)	ctime(S)	same
ultoa(DOS)	ultoa(DOS)	DOS-OS/2 only
umask(DOS)	umask(DOS)	differs under DOS-OS/2
ungetc(DOS)	ungetc(S)	same
ungetch(DOS)	ungetch(DOS)	DOS-OS/2 only

(Continued on next page)

DOS routines/macros
(Continued)

Routine/macros(*)	Manual page	Note
unlink(DOS)	unlink(DOS)	differs under DOS-OS/2
_unregisterfonts(DOS)	_unregisterfonts(DOS)	DOS only
utime(DOS)	utime(DOS)	differs under DOS-OS/2
va_arg(DOS)(*)	varargs(S)	same
va_end(DOS)(*)	varargs(S)	same
va_start(DOS)(*)	varargs(S)	same
vfprintf(DOS)	vprintf(S)	same
vprintf(DOS)	vprintf(S)	same
vsprintf(DOS)	vprintf(S)	same
write(DOS)	write(DOS)	differs under DOS-OS/2
wait(DOS)	wait(DOS)	DOS-OS/2 only
_wrapon(DOS)	_wrapon(DOS)	DOS-OS/2 only
y0(DOS)	bessel(S)	same
_y0(DOS)	_y0(DOS)	DOS-OS/2 only
y1(DOS)	bessel(S)	same
_y1l(DOS)	_y1l(DOS)	DOS-OS/2 only
yn(DOS)	bessel(S)	same
_ynl(DOS)	_ynl(DOS)	DOS-OS/2 only

abort

void abort(void);

Include	PROCESS.H or STDLIB.H
Returns	No return value

Aborts the current process.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

abs

int abs(int *n*);

Include	STDLIB.H or MATH.H
n	Value whose absolute value is sought
Returns	The absolute value of *n*

Calculates the absolute value of its argument.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

access

int access(char ****pathname*, **int *mode*);**

Include	IO.H, ERRNO.H
pathname	File or directory path name
mode	**00** (exist), **02** (write), **04** (read), **06** (read/write)
Returns	0 if the file has the given access mode or if the directory exists; –1 if the directory or file does not exist or does not have the given mode
errno	**EACCESS, ENOENT**

Determines whether a specified file or directory exists, and (in the case of a file) whether it can be accessed in the specified mode.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

acos

double acos(double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arccosine is sought
Returns	0 if *x* is less than –1 or greater than 1
errno	**EDOM**

Returns the arccosine of *x* in the range 0 to pi radians.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

acosl

long double acosl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arccosine is sought
Returns	0 if *x* is less than –1 or greater than 1
errno	**EDOM**

Returns the arccosine of *x* in the range 0 to pi radians. Long double-precision floating-point version of **acos**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

alloca

void *alloca(size_t *size* **);**

Include	MALLOC.H
size	Bytes to be allocated from the stack
Returns	A **void** pointer to the allocated space; **NULL** if the space cannot be allocated

Allocates *size* bytes from the program's stack.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_arc

short _far _arc(short *x1*, **short** *y1*, **short** *x2*, **short** *y2*, **short** *x3*, **short** *y3*, **short** *x4*, **short** *y4* **);**

Include	GRAPH.H
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
x3, *y3*	Start vector
x4, *y4*	End vector
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an elliptical arc using the view coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_arc_w

short _far _arc_w(double *x1*, **double** *y1*, **double** *x2*, **double** *y2*, **double** *x3*, **double** *y3*, **double** *x4*, **double** *y4* **);**

Include	GRAPH.H
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
x3, *y3*	Second point of start vector (center of bounding rectangle is first point)
x4, *y4*	Second point of end vector (center of bounding rectangle is first point)
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an elliptical arc using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_arc_wxy

short _far _arc_wxy(struct _wxycoord _far ****pwxy1*, struct _wxycoord _far ****pwxy2***,**
struct _wxycoord _far ****pwxy3*, struct _wxycoord _far ****pwxy4** **);**

Include	GRAPH.H
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
pwxy3	Start vector
pwxy4	End vector
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an elliptical arc using the window coordinate system.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

asctime

char *asctime(const struct tm ****timeptr* **);

Include	TIME.H
timeptr	Time/date structure defined in TIME.H
Returns	A pointer to the character string result

Converts a value stored as a **tm** time structure to a character string.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

asin

double asin(double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arcsine is sought
Returns	0 if *x* is less than –1 or greater than 1
errno	**EDOM**

Calculates the arcsine of a value in the range –pi/2 to pi/2 radians.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

asinl

long double asinl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arcsine is sought
Returns	0 if *x* is less than –1 or greater than 1
errno	**EDOM**

Calculates the arcsine of a value in the range –pi/2 to pi/2 radians. Long double-precision floating-point version of **asin.**

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

assert

void assert(int *expression* **);**

Include	ASSERT.H, STDIO.H
expression	A logical expression
Returns	No return value

Prints a diagnostic message and calls the **abort** routine if a logical expression is false (0). Activated only if **NDEBUG** is defined at compile time.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atan

double atan(double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arctangent is sought
Returns	The arctangent result; 0 if *x* is 0
errno	**EDOM**

Calculates the arctangent of *x* in the range –pi/2 to pi/2.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atanl

long double atanl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Value whose arctangent is sought
Returns	The arctangent result; 0 if *x* is 0
errno	**EDOM**

Calculates the arctangent of *x* in the range –pi/2 to pi/2. Long double-precision floating-point version of **atan**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

atan2

double atan2(double *y*, **double** *x* **);**

Include	MATH.H, ERRNO.H
y/x	Value whose arctangent is sought
Returns	The arctangent result; 0 if *x* or *y* is 0
errno	**EDOM**

Calculates the arctangent of *y/x* in the range –pi to pi.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atan2l

long double atan2l(long double *y*, **long double** *x* **);**

Include	MATH.H, ERRNO.H
y/x	Value whose arctangent is sought
Returns	The arctangent result; 0 if *x* or *y* is 0
errno	**EDOM**

Calculates the arctangent of *y/x* in the range –pi to pi. Long double-precision floating-point version of **atan2**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

atexit

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

Include	STDLIB.H
func	Function to call on exit
Returns	0 if successful; a nonzero value if not

Calls the specified function at exit. Successive calls create a register of up to 32 functions to be called (last in, first out).

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

atof

double atof(const char ****string* **);

Include	MATH.H or STDLIB.H
string	String to be converted
Returns	The converted string; 0 if the string cannot be converted

Converts a character string to a double-precision floating-point value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atoi

int atoi(const char ****string* **);

Include	STDLIB.H
string	String to be converted
Returns	The converted string; 0 if the string cannot be converted

Converts a character string to an integer value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

atol

long atol(const char ****string* **);

Include	STDLIB.H
string	String to be converted
Returns	The converted string; 0 if the string cannot be converted

Converts a character string to a long integer value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_atold

long double _atold(const char *_string_);

Include	MATH.H or STDLIB.H
string	String to be converted
Returns	The converted string; 0 if the string cannot be converted

Converts a character string to a long double-precision floating-point value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bcalloc

void _based(void) *_bcalloc(_segment *seg*, **size_t** *num*, **size_t** *size* **);**

Include	MALLOC.H
seg	Based-heap segment selector
num	Number of elements
size	Length in bytes of each element
Returns	A based pointer to the allocated space; **_NULLOFF** if there is insufficient memory available, if *num* or *size* is 0, or if the specified segment has not been initialized as part of the based heap

Allocates array storage on the specified based heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

bdos

int bdos(int *dosfunc*, **unsigned int** *dosdx*, **unsigned int** *dosal* **);**

Include	DOS.H
dosfunc	Function number
dosdx	DX register value
dosal	AL register value
Returns	The value of the AX register after the system call has completed

Invokes the DOS system call specified by a function number.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_beginthread

int _far _beginthread(void(_far ***_start_address_)(void _far *),**
void _far ***_stack_bottom_, unsigned** _stack_size_**, void _far ***_arglist_ **);**

Include	PROCESS.H, STDDEF.H, ERRNO.H
start_address	Address where execution begins
stack_bottom	Address of the thread stack or NULL
stack_size	Stack space to reserve
arglist	Address of data item to pass to new thread
Returns	The thread identification number of the new thread if successful; –1 to indicate an error if not
errno	**EINVAL, EAGAIN**

Creates a thread that begins execution at *start_address*.

☐ ANSI ☐ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bexpand

void _based(void) *_bexpand(_segment *seg*, **void _based(void) ***memblock*, **size_t** *size* **);**

Include	MALLOC.H
seg	Based-heap segment selector
memblock	Pointer to previously allocated memory
size	New size in bytes
Returns	A based pointer to the resized memory block if successful; **_NULLOFF** if not

Changes the size of a block in the based heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bfree

void _bfree(_segment *seg*, **void _based(void) ***memblock* **);**

Include	MALLOC.H
seg	Segment selected
memblock	Allocated memory block
Returns	No return value

Frees a memory block in the based heap (previously allocated by a call to **_bmalloc** or **_brealloc**).

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bfreeseg

int _bfreeseg(_segment *seg* **);**

Include	MALLOC.H
seg	Based-heap segment selector
Returns	0 if successful; –1 in the case of an error

Frees a specified based-heap segment.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapadd

int _bheapadd(_segment *seg*, **void _based(void)** ***memblock*, **size_t** *size* **);**

Include	MALLOC.H
seg	Based-heap segment selector
memblock	Pointer to based memory
size	Size in bytes of memory to add
Returns	A segment selector if successful; –1 if not

Adds an unused piece of memory to the specified based heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapchk

int _bheapchk(_segment *seg* **);**

Include	MALLOC.H
seg	Based-heap segment to check or **_NULLSEG** to check all based-heap segments
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks for minimal consistency in the based heap specified by the selector value *seg*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapmin

int _bheapmin(_segment *seg* **);**

Include	MALLOC.H
seg	Based-heap segment to minimize, or **_NULLSEG** for all based-heap segments
Returns	0 if successful; –1 if not

Releases unused memory in the based heap, minimizing the heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapseg

_segment _bheapseg(size_t *size* **);**

Include	MALLOC.H
size	Segment size to allocate
Returns	The selector for the newly allocated segment if successful; –1 if not

Allocates a based-heap segment.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapset

int _bheapset(_segment *seg*, **unsigned int** *fill* **);**

Include	MALLOC.H
seg	Based-heap segment selector or **_NULLSEG** for all based-heap segments
fill	Fill character
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks for minimal consistency in the based-heap segment specified by the selector value *seg*, and then sets the heap's free entries with the *fill* value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bheapwalk

int _bheapwalk(_segment *seg*, **_HEAPINFO** ***entryinfo*);**

Include	MALLOC.H
seg	Based-heap selector or **_NULLSEG** for all based-heap segments
entryinfo	Structure to contain information about the next heap entry, defined in MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADPTR, _HEAPBADBEGIN, _HEAPBADNODE, _HEAPEND**

Walks through the based-heap segment specified by the selector *seg* (or all based-heap segments if *seg* is equal to **_NULLSEG**), one entry per call, returning a pointer to a **_HEAPINFO** structure that contains information about the next based-heap entry.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bios_disk

unsigned _bios_disk(unsigned *service*, **struct diskinfo_t** ***diskinfo*);**

Include	BIOS.H
service	**_DISK_RESET, _DISK_STATUS, _DISK_READ, _DISK_WRITE, _DISK_VERIFY, _DISK_FORMAT**
diskinfo	Disk parameters structure defmed in BIOS.H
Returns	The value stored in the AX register upon return from the BIOS interrupt

Calls BIOS disk-access services (INT 0x13).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_equiplist

unsigned _bios_equiplist(void);

Include BIOS.H

Returns A set of bits indicating what is installed

Calls INT 0x11 to determine what hardware and peripherals are currently installed on the machine.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_keybrd

unsigned _bios_keybrd(unsigned *service* **);**

Include BIOS.H

service **_KEYBRD_READ, _N_KEYBRD_READ, _KEYBRD_READY, _N_KEYBRD_READY, _KEYBRD_SHIFTSTATUS, _N_KEYBRD_SHIFTSTATUS**

Returns The value returned in the AX register upon return from the BIOS interrupt

Calls BIOS keyboard services (INT 0x16).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_memsize

unsigned _bios_memsize(void);

Include BIOS.H

Returns The total amount of installed memory in 1K blocks

Calls BIOS memory-size service (INT 0x12) to determine the total amount of main memory installed.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_printer

unsigned _bios_printer(unsigned *service*, **unsigned** *printer*, **unsigned** *data* **);**

Include	BIOS.H
service	**_PRINTER_WRITE, _PRINTER_INIT, _PRINTER_STATUS**
printer	Target printer port
data	Output data
Returns	The value returned in the AX register upon return from the BIOS interrupt

Calls BIOS printer services (INT 0x17) to perform printer tasks for parallel printers.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_serialcom

unsigned _bios_serialcom(unsigned *service*, **unsigned** *serial_port*, **unsigned** *data* **);**

Include	BIOS.H
service	**_COM_INIT, _COM_RECEIVE, _COM_SEND, _COM_STATUS**
serial_port	Serial port to use
data	**_COM_CHR7, _COM_CHR8, _COM_STOP1, _COM_STOP2, _COM_NOPARITY, _COM_EVENPARITY, _COM_ODDPARITY, _COM_110, _COM_150, _COM_300, _COM_600, _COM_1200, _COM_2400, _COM_4800, _COM_9600**
Returns	A 16-bit integer whose meaning varies depending on the service requested

Calls BIOS communications services (INT 0x14) to provide serial communications.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bios_timeofday

unsigned _bios_timeofday(unsigned *service*, **long ***_timeval_ **);**

Include	BIOS.H
service	**_TIME_GETCLOCK, _TIME_SETCLOCK**
timeval	Clock count
Returns	The value returned in the AX register upon return from the BIOS interrupt

Calls BIOS time and date services (INT 0x1A) to get or set the current system clock count.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_bmalloc

void _based(void) *_bmalloc(_segment *seg*, **size_t** *size* **);**

Include	MALLOC.H
seg	Based-heap segment selector
size	Number of bytes to allocate
Returns	A **void** pointer to the allocated space; **_NULLOFF** if it fails

Allocates a memory block of at least *size* bytes in the based-heap segment specified by the selector *seg*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_bmsize

size_t _bmsize(_segment *seg*, **void _based(void)** ***memblock*);**

Include	MALLOC.H
seg	Based-heap segment selector
memblock	Pointer to based-heap memory block
Returns	The size in bytes

An unsigned integer representing the size in bytes of the based-heap memory block allocated by a call to **_bcalloc**, **_bmalloc**, or **_brealloc**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_brealloc

void _based(void) *_brealloc(_segment *seg*, **void _based(void)** ***memblock*,
size_t *size* **);**

Include	MALLOC.H
seg	Based-heap segment selector
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A **void _based** pointer to the reallocated memory block if successful; **_NULLOFF** if not

Changes the size of a previously allocated based-heap memory block, possibly relocating it.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

bsearch

void *bsearch(const void ***_key_, const void ***_base_**, size_t** _num_**, size_t** _width_**,**
int (***_compare_ **)(const void ***_elem1_, const void ***_elem2_ **));**

Include	STDLIB.H, SEARCH.H
key	Pointer to object to search for
base	Pointer to base of search data
num	Number of elements
width	Width of elements
compare	Comparison function
elem1	Pointer to the key for the search
elem2	Pointer to the array element to be compared with the key
Returns	A pointer to the first occurrence of the object pointed to by *key* in the array that *base* points to; **NULL** if *key* is not found

Performs a binary search of a sorted array. Note that *key* is a pointer to an object of size *width*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cabs

double cabs(struct complex *z* **);**

Include	MATH.H, ERRNO.H
z	Complex number structure defined in MATH.H
Returns	**HUGE_VAL** on overflow
errno	**ERANGE**

Calculates the absolute value of a complex number.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cabsl

long double cabsl(struct _complexl *z* **);**

Include	MATH.H, ERRNO.H
z	Long double-precision complex number structure defined in MATH.H
Returns	The absolute value if successful; **_LHUGE_VAL** on overflow
errno	**ERANGE**

Calculates the absolute value of a complex number. Long double-precision floating-point version of **cabs**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

calloc

void *calloc(size_t *num*, **size_t** *size* **);**

Include	STDLIB.H or MALLOC.H
num	Number of elements
size	Length in bytes of each element
Returns	A pointer to the allocated space; **NULL** if there is insufficient memory available

Allocates storage for an array of *num* elements of *size* bytes. All bytes are initialized to 0.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ceil

double ceil(double *x*);

Include	MATH.H
x	Floating-point value
Returns	A value representing the smallest integer that is greater than or equal to the argument

Calculates the ceiling of a value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ceill

long double ceill(long double *x*);

Include	MATH.H
x	Long double-precision floating-point value
Returns	The long double result, rounded up

Returns a long double-precision floating-point value representing the smallest integer that is greater than or equal to the argument.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_cexit

void _cexit(void);

Include	PROCESS.H
Returns	No return value

Performs the **exit** termination procedures (e.g., flushing buffers) but returns control to the caller; does not terminate the process.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_c_exit

void _c_exit(void);

Include	PROCESS.H
Returns	No return value

Performs **_exit** termination procedures (i.e., without cleanup operations), but returns control to the caller; does not terminate the process.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

cgets

char *cgets(char *_buffer_);

Include	CONIO.H
buffer	Storage location for data
Returns	A pointer to the start of a string of characters

Reads a string of characters (excluding terminal **\n**) directly from the console and stores the string and its length in *buffer*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_chain_intr

void _chain_intr(void(_interrupt _far *_target_)( ));

Include	DOS.H
target	Target interrupt routine
Returns	No return value

Chains one interrupt handler to another interrupt handler.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

chdir

int chdir(char ***_dirname_);**

Include	DIRECT.H, ERRNO.H
dirname	Path name of new working directory
Returns	0 if the working directory is successfully changed; –1 if the specified path name could not be found
errno	**ENOENT**

Changes the current working directory.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_chdrive

int _chdrive(int *drive* **);**

Include	DIRECT.H
drive	Number of new working drive (A: = 1, B: = 2)
Returns	0 if the working drive is successfully changed; –1 if not

Changes the current working drive.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

chmod

int chmod(char ***_filename_, int** _pmode_ **);**

Include	SYS\TYPES.H, SYS\STAT.H, IO.H, ERRNO.H
filename	Path name of existing file
pmode	Permission setting for file
Returns	0 if the permission setting is successfully changed; **ENOENT** if the specified file could not be found
errno	**ENOENT**

Changes file permission settings.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

chsize

int chsize(int _handle_**, long** _size_ **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
size	New length of file in bytes
Returns	0 if the file size is successfully changed; –1 if not
errno	**EACCESS, EBADF, ENOSPC**

Changes the size of a file.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_clear87

unsigned int _clear87(void);

Include	FLOAT.H
Returns	A word whose bits indicate the floating-point status

Gets and clears a floating-point status word.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

clearerr

void clearerr(FILE *_stream_);

Include	STDIO.H
stream	Pointer to **FILE** structure
Returns	No return value

Resets the error and end-of-file indicators for a stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_clearscreen

void _far _clearscreen(short *area* **);**

Include	GRAPH.H
area	**_GCLEARSCREEN, _GVIEWPORT, _GWINDOW**
Returns	No return value

Clears the screen and fills it with the current background color.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

clock

clock_t clock(void);

Include	TIME.H
Returns	The elapsed processor time; –1, cast as **clock_t**, if the processor time is not available

Calculates how much processor time in seconds has been used by the calling process.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

close

int close(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
Returns	0 if the file was successfully closed; –1 if the file-handle argument is invalid
errno	**EBADF**

Closes a file.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_control87

unsigned int _control87(unsigned int *new*, **unsigned int** *mask* **);**

Include	FLOAT.H
new	New control-word bit values
mask	Mask for new control-word bits to set
Returns	A word whose bits indicate the floating-point control state

Gets and sets the floating-point control word.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

cos

double cos(double *x* **);**

Include	MATH.H, ERRNO.H
x	Radians
Returns	The cosine of *x*; 0 if *x* is so large that significance is completely lost
errno	**ERANGE**

Calculates the cosine of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cosh

double cosh(double *x* **);**

Include	MATH.H, ERRNO.H
x	Radians
Returns	The hyperbolic cosine of *x*; 0 if *x* is so large that significance is completely lost; **HUGE_VAL** if the result is too large
errno	**ERANGE**

Calculates the hyperbolic cosine of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

coshl

long double coshl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Radians
Returns	The hyperbolic cosine of *x* if successful; 0 if *x* is so large that significance is completely lost; **_LHUGE_VAL** if the result is too large
errno	**ERANGE**

Calculates the hyperbolic cosine of *x*. Long double-precision floating-point version of **cosh**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

cosl

long double cosl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Radians
Returns	The cosine of *x* if successful; 0 if *x* is so large that significance is completely lost
errno	**ERANGE**

Calculates the cosine of *x*. Long double-precision floating-point version of **cos**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

cprintf

int cprintf(char ***_format_ [[, _argument_]]... **);

Include	CONIO.H
format	Format-control string
argument	Optional arguments
Returns	The number of characters printed

Formats and prints to the console. See **printf/scanf** format table.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

cputs

int cputs(char ***_string_ **);

Include	CONIO.H
string	Output string
Returns	0 if successful; a nonzero value if not

Writes a null-terminated string directly to the console.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

creat

int creat(char ***_filename_, int** _pmode_ **);**

Include	SYS\TYPES.H, SYS\STAT.H, IO.H, ERRNO.H
filename	Path name of new file
pmode	**S_IWRITE, S_IREAD, S_IREAD|S_IWRITE**
Returns	A handle for the created file if successful; –1 if not
errno	**EACCESS, EMFILE, ENOENT**

Creates a new file or opens and truncates an existing file.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cscanf

int cscanf(char ***_format_ [[, _argument_]]... **);

Include	CONIO.H
format	Format-control string
argument	Optional arguments
Returns	The number of fields that were successfully converted and assigned; **EOF** for an attempt to read at end-of-file

Reads formatted data directly from the console into a specified location. See **printf/scanf** format table.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

ctime

char *ctime(const time_t *_timer_);

Include	TIME.H
timer	Pointer to stored time
Returns	A pointer to the character string result; **NULL** if *timer* represents a date before 1980

Converts time stored as a **time_t** value to a character string.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

cwait

int cwait(int *_termstat_, int _procid_, int _action_);

Include	PROCESS.H, ERRNO.H
termstat	Termination status and return code
procid	Child process identification
action	**WAIT_CHILD, WAIT_GRANDCHILD**
Returns	The child process identification after normal termination; –1 otherwise
errno	**EINTR, EINVAL, ECHILD**

Suspends the calling process until the child process terminates, and updates the *termstat* variable.

□ ANSI □ DOS ■ OS/2 □ UNIX □ XENIX

dieeetomsbin

int dieeetomsbin(double ****src8*, double ****dst8** **);**

Include	MATH.H
src8	Buffer containing number to convert
dst8	Buffer for converted number
Returns	0 if the conversion is successful; 1 if the conversion causes an overflow

Converts a double-precision number in IEEE format to Microsoft binary format.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

difftime

double difftime(time_t ***timer1*****, time_t** ***timer0*** **);**

Include	TIME.H
timer1, *timer0*	Time value structures defined in TIME.H
Returns	The difference between two times

Computes the elapsed time in seconds between two time values.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_disable

void _disable(void);

Include	DOS.H
Returns	No return value

Disables interrupts by executing an 8086 CLI machine instruction.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_displaycursor

short _far _displaycursor(short *toggle* **);**

Include	GRAPH.H
toggle	**_GCURSOROFF, _GCURSORON**
Returns	The previous value of *toggle*

Determines whether the cursor will be displayed.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

div

div_t div(int *numer*, **int** *denom* **);**

Include	STDLIB.H
numer	Numerator
denom	Denominator
Returns	The quotient and remainder in a structure defined in STDLIB.H

Divides the numerator by the denominator computing the quotient and remainder.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

dmsbintoieee

int dmsbintoieee(double ***_src8_, double ***_dst8_ **);**

Include	MATH.H
src8	Buffer containing number to convert
dst8	Buffer for converted number
Returns	0 if the conversion is successful; 1 if the conversion causes an overflow

Converts a double-precision number in Microsoft binary format to IEEE format.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_dos_allocmem

unsigned _dos_allocmem(unsigned *size*, unsigned **seg*);

Include	DOS.H, ERRNO.H
size	Block size to allocate
seg	Segment descriptor return buffer
Returns	0 if successful; the DOS error code if not
errno	**ENOMEM**

Allocates a block of memory *size* paragraphs long (16 bytes each).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_close

unsigned _dos_close(int *handle*);

Include	DOS.H, ERRNO.H
handle	Target file handle
Returns	0 if successful; the DOS error code if not
errno	**EBADF**

Uses system call 0x3E to close a file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_creat

unsigned _dos_creat(char *_filename_, unsigned _attrib_, int *_handle_);

Include	DOS.H, ERRNO.H
filename	File path name
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
handle	Handle return buffer
Returns	0 if successful; the DOS error code if not
errno	**ENOENT, EMFILE, EACCESS, EEXIST**

Uses system call 0x3C to create a new file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_creatnew

unsigned _dos_creatnew(char *_filename_, unsigned _attrib_, int *_handle_);

Include	DOS.H, ERRNO.H
filename	File path name
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
handle	Handle return buffer
Returns	0 if successful; the DOS error code if not
errno	**ENOENT, EMFILE, EACCESS, EEXIST**

Uses system call 0x5B to create a new file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

dosexterr

int dosexterr(struct DOSERROR ****errorinfo* **);

Include	DOS.H
errorinfo	Error-information structure defined in DOS.H
Returns	The value in the AX register (identical to the value in the **exterror** structure field)

Uses system call 0x59 to get extended error information.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_findfirst

unsigned _dos_findfirst(char ****filename*, **unsigned *attrib*, **struct find_t ****fileinfo* **);**

Include	DOS.H, ERRNO.H
filename	Target file name
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
fileinfo	File-information return structure defined in DOS.H
Returns	0 if successful; the DOS error code if not
errno	**ENOENT**

Uses system call 0x4E to return information about the first instance of a file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_findnext

unsigned _dos_findnext(struct find_t *_fileinfo_);

Include	DOS.H, ERRNO.H
fileinfo	File-information return structure defined in DOS.H
Returns	0 if successful; the DOS error code if not
errno	**ENOENT**

Uses system call 0x4F to find the next file that matches the name and attributes specified in a prior call to **_dos_findfirst**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_freemem

unsigned _dos_freemem(unsigned _seg_);

Include	DOS.H, ERRNO.H
seg	Block to be released
Returns	0 if successful; the DOS error code if not
errno	**ENOMEM**

Uses system call 0x49 to release a block of memory previously allocated by **_dos_allocmem.**

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getdate

void _dos_getdate(struct dosdate_t *_date_);

Include	DOS.H
date	Current date structure defined in DOS.H
Returns	No return value

Uses system call 0x2A to get the current system date.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getdiskfree

unsigned _dos_getdiskfree(unsigned *drive*, **struct diskfree_t ***_diskspace_ **);**

Include	DOS.H, ERRNO.H
drive	Target disk drive
diskspace	Disk-information structure defined in DOS.H
Returns	0 if successful; a nonzero value if not
errno	**EINVAL**

Uses system call 0x36 to get information on a disk drive.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getdrive

void _dos_getdrive(unsigned ***_drive_ **);

Include	DOS.H
drive	Current drive return buffer
Returns	No return value

Uses system call 0x19 to get the current disk drive (A: = 0, B: = 1).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getfileattr

unsigned _dos_getfileattr(char ****pathname*, unsigned ****attrib** **);**

Include	DOS.H, ERRNO.H
pathname	Full path of target file or directory
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
Returns	0 if successful; the DOS error code if not
errno	**ENOENT**

Uses system call 0x43 to get the current attributes of a file or directory.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getftime

unsigned _dos_getftime(int ***handle*****, unsigned ****date***, unsigned ****time** **);**

Include	DOS.H, ERRNO.H
handle	Target file
date	Date-return buffer
time	Time-return buffer
Returns	0 if successful; the DOS error code if not
errno	**EBADF**

Uses system call 0x57 to get the time and date a file was last written.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_gettime

void _dos_gettime(struct dostime_t *_time_);

Include	DOS.H
time	Current system time structure defined in DOS.H
Returns	No return value

Uses system call 0x2C to get the current system time.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_getvect

void(_interrupt _far *_dos_getvect(unsigned _intnum_))( );

Include	DOS.H
intnum	Target interrupt vector
Returns	Far pointer to the current interrupt handler

Uses system call 0x35 to get the current value of an interrupt vector.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_keep

void _dos_keep(unsigned _retcode_, unsigned _memsize_);

Include	DOS.H
retcode	Exit status code
memsize	Allocated resident memory in 16-byte paragraphs
Returns	No return value

Uses system call 0x31 to install a terminate-and-stay-resident (TSR) program in memory.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_open

unsigned _dos_open(char ****filename*, unsigned** *mode***, int ****handle* **);**

Include	DOS.H, FCNTL.H, SHARE.H, ERRNO.H
filename	Path name to an existing file
mode	**O_RDONLY, O_WRONLY, O_RDWR, SH_COMPAT, SH_DENYRW, SH_DENYWR, SH_DENYRD, SH_DENYNO, O_NOINHERIT**
handle	Handle-return buffer
Returns	0 if successful; the DOS error code if not
errno	**EINVAL, ENOENT, EMFILE, EACCESS**

Uses system call 0x3D to open an existing file, with modes specified by *mode*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_read

unsigned _dos_read(int *handle***, void _far ****buffer***, unsigned** *count***, unsigned ****numread* **);**

Include	DOS.H, ERRNO.H
handle	File to read
buffer	Buffer to write to
count	Number of bytes to read
numread	Number of bytes actually read
Returns	0 if successful; the DOS error code if not
errno	**EACCESS, EBADF**

Uses system call 0x3F to read data from a file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setblock

unsigned _dos_setblock(unsigned *size*, **unsigned** *seg*, **unsigned** *****_maxsize_ **);**

Include	DOS.H, ERRNO.H
size	New segment size
seg	Target segment
maxsize	Maximum size buffer
Returns	0 if successful; the DOS error code if not
errno	**ENOMEM**

Changes the size of a segment previously allocated by **_dos_allocmem.**

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setdate

unsigned _dos_setdate(struct dosdate_t *****_date_ **);**

Include	DOS.H, ERRNO.H
date	System date structure defined in DOS.H
Returns	0 if successful; a nonzero value if not
errno	**EINVAL**

Uses system call 0x2B to set the current system date.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setdrive

void _dos_setdrive(unsigned *drive*, **unsigned ***_numdrives_ **);**

Include	DOS.H
drive	New default drive
numdrives	Total drives available
Returns	No return value

Uses system call 0x0E to set the current default drive.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setfileattr

unsigned _dos_setfileattr(char ***_pathname_, **unsigned *attrib* **);**

Include	DOS.H, ERRNO.H
pathname	Full path of target file or directory
attrib	**_A_ARCH, _A_HIDDEN, _A_NORMAL, _A_RDONLY, _A_SUBDIR, _A_SYSTEM, _A_VOLID**
Returns	0 if successful; the DOS error code if not
errno	**EACCESS, ENOENT**

Uses system call 0x43 to set the attributes of a file or a directory.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setftime

unsigned _dos_setftime(int *handle*, **unsigned** *date*, **unsigned** *time* **);**

Include	DOS.H, ERRNO.H
handle	Target file
date	Date of last write
time	Time of last write
Returns	0 if successful; the DOS error code if not
errno	**EBADF**

Uses system call 0x57 to set the date and time a file was last written.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_settime

unsigned _dos_settime(struct dostime_t ***_time_ **);

Include	DOS.H, ERRNO.H
time	System time structure defined in DOS.H
Returns	0 if successful; a nonzero value if not
errno	**EINVAL**

Uses system call 0x2D to set the current system time.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_setvect

void _dos_setvect(unsigned *intnum*, **void(_interrupt _far ***_handler_ **)( ));**

Include	DOS.H
intnum	Target interrupt vector
handler	Interrupt handler to assign
Returns	No return value

Uses system call 0x25 to set the current value of an interrupt vector.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_dos_write

unsigned _dos_write(int *handle*, **void _far ***_buffer_, **unsigned** *count*, **unsigned ***_numwrt_ **);**

Include	DOS.H, ERRNO.H
handle	File to write to
buffer	Buffer to write from
count	Number of bytes to write
numwrt	Number of bytes actually written
Returns	0 if successful; the DOS error code if not
errno	**EACCESS, EBADF**

Uses system call 0x40 to write data to a file.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

dup

int dup(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Handle of an open file
Returns	A new file handle if successful; –1 if not
errno	**EBADF, EMFILE**

Associates a second file handle with the currently open file.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

dup2

int dup2(int *handle1*, **int** *handle2* **);**

Include	IO.H, ERRNO.H
handle1	Handle of an open file
handle2	Any handle value
Returns	0 if successful; –1 if not
errno	**EBADF, EMFILE**

Associates the specified second handle with the currently open file.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ecvt

char *ecvt(double *value*, **int** *count*, **int ***dec*, **int ***sign* **);**

Include	STDLIB.H
value	Number to be converted
count	Number of digits stored
dec	Stored decimal-point position
sign	Sign of converted number
Returns	A pointer to the string of digits

Converts a floating-point number to a character string.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_ellipse

short _far _ellipse(short *control*, **short** *x1*, **short** *y1*, **short** *x2*, **short** *y2* **);**

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an ellipse using view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_ellipse_w

short _far _ellipse_w(short *control*, **double** *wx1*, **double** *wy1*, **double** *wx2*, **double** *wy2* **);**

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
wx1, *wy1*	Upper left corner of bounding rectangle
wx2, *wy2*	Lower right corner of bounding rectangle
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws an ellipse using window coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_ellipse_wxy

short _far _ellipse_wxy(short *control*, **struct _wxycoord _far ***_pwxy1_,
struct _wxycoord _far ***_pwxy2_ **);

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
Returns	A nonzero value if successful; 0 if not

Draws an ellipse using window coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_enable

void _enable(void);

Include	DOS.H
Returns	No return value

Enables interrupts by executing an 8086 STI machine instruction.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_endthread

void _far _endthread(void);

Include	PROCESS.H
Returns	No return value

Terminates a thread created by **_beginthread.**

☐ ANSI ☐ DOS ■ OS/2 ☐ UNIX ☐ XENIX

eof

int eof(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
Returns	1 if the current position is end-of-file; 0 if not; –1 if an error has occurred
errno	**EBADF**

Determines whether the end of the file has been reached.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

execl

int execl(char ****cmdname*, **char ****arg0*, ... **char ****argn*, **NULL);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed individually.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execle

int execle(char ****cmdname*, **char ****arg0*, ... **char ****argn*, **NULL, char *****envp* **);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
envp	Array of pointers to environment settings
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed individually. An array of pointers to environment arguments is explicitly passed to the child process.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execlp

int execlp(char ****cmdname*, **char ****arg0*, ... **char ****argn*, **NULL);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed individually. Uses the PATH environment variable to find the file to be executed.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execlpe

int execlpe(char ****cmdname*, **char ****arg0*, ... **char ****argn*, **NULL, char *****envp* **);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
envp	Array of pointers to environment settings
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed individually. Uses the PATH environment variable to find the file to be executed. An array of pointers to environment arguments is explicitly passed to the child process.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execv

int execv(char ****cmdname*, **char *****argv* **);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed as an array of pointers.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execve

int execve(char ****cmdname*, **char *****argv*, **char *****envp* **);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
envp	Array of pointers to environment settings
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed as an array of pointers. An array of pointers to environment arguments is explicitly passed to the child process.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execvp

int execvp(char ****cmdname*, **char *****argv*);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed as an array of pointers. Uses the PATH environment variable to find the file to be executed.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

execvpe

int execvpe(char ****cmdname*, **char *****argv*, **char *****envp*);

Include	PROCESS.H, ERRNO.H
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
envp	Array of pointers to environment settings
Returns	Does not normally return to the calling process, but if it does, an error has occurred and the return value is –1
errno	**E2BIG, EACCESS, EMFILE, ENOENT, ENOEXEC, ENOMEM**

Loads and executes a new child process, terminating the calling process. Command-line arguments are passed as an array of pointers. Uses the PATH environment variable to find the file to be executed. An array of pointers to environment arguments is explicitly passed to the child process.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

exit

void exit(int *status* **);**

Include	PROCESS.H or STDLIB.H
status	Exit status
Returns	No return value

Terminates a process after normal cleanup operations (e.g., flushing buffers).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_exit

void _exit(int *status* **);**

Include	PROCESS.H or STDLIB.H
status	Exit status
Returns	No return value

Terminates a process without normal cleanup operations.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

exp

double exp(double *x* **);**

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	Exponential function of *x* if successful; 0 on underflow; **HUGE_VAL** on overflow
errno	**ERANGE**

Returns the exponential function of the floating-point argument *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_expand

void *_expand(void *_memblock_, size_t _size_);

Include	MALLOC.H
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A pointer to the resized memory block; **NULL** if there is insufficient memory to expand the block

Changes the size of a previously allocated memory block by attempting to expand or contract the block without moving its location in the heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

expl

long double expl(long double _x_);

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	Exponential function of *x* if successful; 0 on underflow; **_LHUGE_VAL** on overflow
errno	**ERANGE**

Returns the exponential function of the floating-point argument *x*. Long double-precision floating-point version of **exp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fabs

double fabs(double *x*);

Include	MATH.H
x	Floating-point value
Returns	The absolute value of *x*

Gets the absolute value of a double-precision floating-point value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fabsl

long double fabsl(long double *x*);

Include	MATH.H
x	Long double-precision floating-point value
Returns	The absolute value of *x*

Gets the absolute value of a long double-precision floating-point value.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fcalloc

void _far *_fcalloc(size_t *num*, size_t *size*);

Include	MALLOC.H
num	Number of elements
size	Length in bytes of each element
Returns	A far pointer to the allocated space if successful; NULL if there is insufficient memory available

Allocates storage space for an array in the far heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fclose

int fclose(FILE ****stream* **);

Include	STDIO.H
stream	Target stream
Returns	0 if the stream is successfully closed; **EOF** if not

Closes an open stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fcloseall

int fcloseall(void);

Include	STDIO.H
Returns	The total number of streams closed; **EOF** if an error occurs

Closes all open streams.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fcvt

char *fcvt(double *value*, **int** *count*, **int ***dec*, **int ***sign* **);**

Include	STDLIB.H
value	Number to be converted
count	Number of digits after decimal point
dec	Pointer to stored decimal-point position
sign	Pointer to stored sign indicator
Returns	A pointer to the string of digits

Converts a floating-point number to a character string.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fdopen

FILE *fdopen(int *handle*, **char** **mode* **);**

Include	STDIO.H
handle	Handle of open file
mode	Access permissions: **"r", "w", "a", "r+", "w+", "a+", "t", "b"**
Returns	A pointer to the open stream if successful; **NULL** if not

Associates a stream with a file handle, allowing a file opened for low-level I/O to be buffered and formatted.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

feof

int feof(FILE **stream* **);**

Include	STDIO.H
stream	Pointer to a stream
Returns	A nonzero value after the first read operation that attempts to read past the end of the file; 0 if the current position is not end-of-file

Determines whether the end of *stream* has been reached.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ferror

int ferror(FILE **stream* **);**

Include	STDIO.H
stream	Pointer to a stream
Returns	A nonzero value if an error occurred; 0 if not

Tests for a reading or writing error on *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_fexpand

void _far *_fexpand(void _far *_memblock_, size_t _size_);

Include	MALLOC.H
memblock	Pointer to previously allocated memory
size	New size in bytes
Returns	A far pointer to the resized memory block if successful; **NULL** if not

Changes the size of a block in the far heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fflush

int fflush(FILE *_stream_);

Include	STDIO.H
stream	Pointer to target stream
Returns	0 if successful or where the specified stream has no buffer or is open for reading only; **EOF** otherwise

Writes the contents of the buffer associated with the stream to the associated file.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_ffree

void _ffree(void _far *_memblock_);

Include	MALLOC.H
memblock	Allocated memory block in the far heap
Returns	No return value

Frees a memory block in the far heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fgetc

int fgetc(FILE *_stream_);

Include	STDIO.H
stream	Pointer to target stream
Returns	The character read; **EOF** if an error occurred or at end-of-file

Reads a character from *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fgetchar

int fgetchar(void);

Include	STDIO.H
Returns	The character read; **EOF** if an error occurred or at end-of-file

Reads a character from the standard input stream, **stdin**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fgetpos

int fgetpos(FILE *_stream_, fpos_t *_pos_);

Include	STDIO.H, ERRNO.H
stream	Pointer to target stream
pos	Position-indicator buffer
Returns	0 if successful; a nonzero value if not
errno	**EBADF, EINVAL**

Gets the current value of the *stream* file-position indicator.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fgets

char *fgets(char *_string_, int _n_, FILE *_stream_);

Include	STDIO.H
string	Storage location for data
n	Maximum number of characters read and stored
stream	Pointer to target stream
Returns	*string* if successful; **NULL** to indicate an error or end-of-file

Reads a string of, at most, *n* characters from the input *stream* and stores it in *string*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_fheapchk

int _fheapchk(void);

Include	MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks for minimal consistency in the far heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fheapmin

int _fheapmin(void);

Include	MALLOC.H
Returns	0 if successful; –1 if not

Releases unused memory in the far heap back to the operating system.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fheapset

int _fheapset(unsigned int *fill* **);**

Include	MALLOC.H
fill	Fill character
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks for minimal consistency in the far heap and then sets the heap's free entries with the *fill* value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fheapwalk

int _fheapwalk(_HEAPINFO ***entryinfo* **);**

Include	MALLOC.H
entryinfo	Structure to contain information about the next far heap entry, defined in MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADPTR, _HEAPBADBEGIN, _HEAPBADNODE, _HEAPEND**

Walks through the far heap, one entry per call, returning a pointer to a **_HEAPINFO** structure that contains information about the next far heap entry.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fieeetomsbin

int fieeetomsbin(float ****src4*, float ****dst4** **);**

Include	MATH.H
src4	Value to convert
dst4	Buffer for converted value
Returns	0 if successful; 1 if the conversion causes an overflow

Converts a single-precision floating-point number in IEEE format to Microsoft binary format.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

filelength

long filelength(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Target file handle
Returns	The file length in bytes if successful; –1 if not
errno	**EBADF**

Gets the length in bytes of a file.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fileno

int fileno(FILE ****stream* **);

Include	STDIO.H
stream	Pointer to target stream
Returns	The file handle

Returns the file handle currently associated with *stream*.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_floodfill

short _far _floodfill(short *x*, **short** *y*, **short** *boundary* **);**

Include	GRAPH.H
x, *y*	Starting point
boundary	Fill boundary color
Returns	A nonzero value if successful; 0 if the fill could not be completed, the starting point lies on the *boundary* color, or the starting point lies outside the clipping region

Fills an area of the display beginning at the view coordinate point using the current color and fill mask.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_floodfill_w

short _far _floodfill_w(double *wx*, **double** *wy*, **short** *boundary* **);**

Include	GRAPH.H
wx, *wy*	Starting point
boundary	Fill boundary color
Returns	A nonzero value if successful; 0 if the fill could not be completed, the starting point lies on the *boundary* color, or the starting point lies outside the clipping region

Fills an area of the window using the current color and fill mask.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

floor

double floor(double *x*);

Include	MATH.H
x	Floating-point value
Returns	The floating-point result

Returns a floating-point value representing the largest integer that is less than or equal to *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

floorl

long double floorl(long double *x*);

Include	MATH.H
x	Long double-precision floating-point value
Returns	Long double-precision floating-point result, rounded down

Returns a long double-precision floating-point value representing the largest integer that is less than or equal to *x*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

flushall

int flushall(void);

Include	STDIO.H
Returns	The number of open streams

Writes the contents of all buffers for open output streams to their files and clears all buffers for open input streams of their contents.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fmalloc

void _far *_fmalloc(size_t *size*);

Include	MALLOC.H
size	Number of bytes to allocate
Returns	A far pointer to the allocated space; **NULL** if it fails

Allocates a memory block of at least *size* bytes in the far heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemccpy

void _far * _far _fmemccpy(void _far *dest, void _far *src, int c, unsigned int count);

Include	STRING.H or MEMORY.H
dest	Far pointer to destination
src	Far pointer to source
c	Last character to copy
count	Number of characters
Returns	A far pointer to the byte in *dest* that immediately follows the copied character (*c*) if successful; **NULL** if *count* bytes copied before *c* was found

Copies zero or more far bytes of *src* to *dest*, copying up to and including the first occurrence of the character *c* or until *count* bytes have been copied, whichever comes first. Model-independent version of **memccpy**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemchr

void _far * _far _fmemchr(const void _far *_buf, int _c, size_t _count_);

Include	STRING.H or MEMORY.H
buf	Far pointer to buffer
c	Character to copy
count	Number of characters
Returns	A far pointer to the first location of *c* in *buf* if successful; **NULL** if not

Looks for the first occurrence of *c* in the first *count* bytes of *buf*. Model-independent version of **_memchr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemcmp

int _far _fmemcmp(const void _far *_buf1_, const void _far *_buf2_, size_t _count_);

Include	STRING.H or MEMORY.H
buf1, *buf2*	Far pointers to buffers to compare
count	Number of characters
Returns	A value less than, equal to, or greater than 0, depending on whether the size of the buffer pointed to by *buf1* is less than, equal to, or greater than the size of the buffer pointed to by *buf2*

Compares the first *count* bytes of two buffers and returns a value indicating their relationship. Model-independent version of **memcmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemcpy

void _far * _far _fmemcpy(void _far *_dest_, const void _far *_src_, size_t _count_);

Include	STRING.H or MEMORY.H
dest	Far pointer to the destination buffer
src	Far pointer to the buffer to copy from
count	Number of characters to copy
Returns	*dest*

Copies *count* bytes of *src* to *dest*. Model-independent version of **memcpy**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemicmp

int _far _fmemicmp(void _far *_buf1_, void _far *_buf2_, unsigned int _count_);

Include	STRING.H or MEMORY.H
buf1, *buf2*	Far pointers to buffers to compare
count	Number of characters
Returns	A value less than, equal to, or greater than 0, depending on whether the size of the buffer pointed to by *buf1* is less than, equal to, or greater than the size of the buffer pointed to by *buf2*

Compares a specified number of characters from two buffers without regard to letter case. Model-independent version of **memicmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fmemmove

void _far * _far _fmemmove(void _far *_dest_, const void _far *_src_, size_t _count_);

Include	STRING.H
dest	Far pointer to destination buffer
src	Far pointer to source buffer
count	Number of characters to copy
Returns	The value of *dest*

Copies *count* characters from *src* to *dest*. Model-independent version of **memmove**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fmemset

void _far * _far _fmemset(void _far *_dest_, int _c_, size_t _count_);

Include	STRING.H or MEMORY.H
dest	Far pointer to destination
c	Character to set
count	Number of characters
Returns	A far pointer to *dest*

Sets the first *count* bytes of *dest* to the character *c*. Model-independent version of **memset**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fmod

double fmod(double *x*, **double** *y* **);**

Include	MATH.H
x, *y*	Floating-point values
Returns	The floating-point remainder

Calculates the floating-point remainder *f* of (*x*, *y*) such that $x = i * y + f$, where *i* is an integer, *f* has the same sign as *x*, and the absolute value of *f* is less than the absolute value of *y*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fmodl

long double fmodl(long double *x*, **long double** *y* **);**

Include	MATH.H
x, *y*	Floating-point values
Returns	The floating-point remainder

Calculates the long double-precision floating-point remainder *f* of (*x*, *y*) such that $x = i * y + f$, where *i* is an integer, *f* has the same sign as *x*, and the absolute value of *f* is less than the absolute value of *y*. Long double-precision floating-point version of **fmod**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fmsbintoieee

int fmsbintoieee(float ****src4*, **float ****dst4* **);

Include	MATH.H
src4	Value to convert
dst4	Buffer for converted value
Returns	0 if successful; 1 if the conversion causes an overflow

Converts a floating-point number in Microsoft binary format to IEEE format.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fmsize

size_t _fmsize(void _far ****memblock* **);

Include	MALLOC.H
memblock	Pointer to far memory block
Returns	An unsigned integer representing the size in bytes of the far-heap memory block allocated by a call to **_fcalloc**, **_fmalloc**, or **_frealloc**

Returns the size of the memory block pointed to by *memblock*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fopen

FILE *fopen(const char ****filename*, const char ****mode* **);**

Include	STDIO.H
filename	Path name of file
mode	Access permissions: **"r"**, **"r+"**, **"r+b"**, **"r+t"**, **"w"**, **"w+"**, **"w+b"**, **"w+t"**, **"a"**, **"a+"**, **"a+b"**, **"a+t"**
Returns	A pointer to the open file if successful; **NULL** if not

Opens a specified file for stream I/O.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

FP_OFF

unsigned FP_OFF(void _far ****address* **);

Include	DOS.H
address	Far pointer to memory address
Returns	An offset

Sets or gets the offset portion of a far address.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fpreset

void _fpreset(void);

Include	FLOAT.H
Returns	No return value

Reinitializes the floating-point math package.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fprintf

int fprintf(FILE ****stream*, **const char ****format* [[, *argument*]]...);

Include	STDIO.H
stream	Pointer to target stream
format	Format-control string
argument	Optional arguments
Returns	The number of characters printed if successful; a negative value if not

Formats and prints a series of characters and values to the output *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

FP_SEG

unsigned FP_SEG(void _far ****address*);

Include	DOS.H
address	Far pointer to memory address
Returns	A segment

Sets or gets the segment selector portion of a far address.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fputc

int fputc(int *c*, **FILE** ***stream*);**

Include	STDIO.H
c	Character to be written
stream	Pointer to target stream
Returns	The character written if successful; **EOF** if not

Writes a single character to the target stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fputchar

int fputchar(int *c* **);**

Include	STDIO.H
c	Character to be written
Returns	The character written if successful; **EOF** if not

Writes a single character to the standard output stream, **stdout.**

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fputs

int fputs(const char ***string*, FILE** ***stream*);**

Include	STDIO.H
string	Output string
stream	Pointer to target stream
Returns	A nonnegative value if successful; **EOF** if not

Copies a string to the target stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fread

size_t fread(void ****buffer*, size_t** ***size*****, size_t** ***count*****, FILE ****stream* **);**

Include	STDIO.H
buffer	Storage location for data
size	Item size in bytes
count	Maximum number of items to be read
stream	Pointer to target stream
Returns	The number of items actually read, which may be less than *count* if an error occurs

Reads up to *count* items of *size* bytes from the input *stream* and stores them in *buffer*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_frealloc

void _far *_frealloc(void _far ****memblock*, size_t** ***size*** **);**

Include	MALLOC.H
memblock	Pointer to previously allocated far memory block
size	New size in bytes
Returns	A far pointer to the reallocated memory block if successful; **NULL** if not

Changes the size of a previously allocated far memory block, possibly moving it.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

free

void free(void ***_memblock_);**

Include	STDLIB.H, MALLOC.H
memblock	Allocated memory block
Returns	No return value

Frees a memory block previously allocated by **calloc**, **malloc**, or **realloc**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_freect

unsigned int _freect(size_t *size* **);**

Include	MALLOC.H
size	Item size in bytes
Returns	The number of items that can be allocated in the near heap

Determines how much memory is available for dynamic memory allocation in the near heap by returning the number of times an item can be allocated.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

freopen

FILE *freopen(const char *_filename_, const char *_mode_, FILE *_stream_);

Include	STDIO.H
filename	Path name of new file
mode	Access permission: **"r", "r+", "r+b" , "r+t", "w", "w+", "w+b", "w+t", "a", "a+", "a+b", "a+t"**
stream	Pointer to target stream
Returns	A pointer to the newly opened file if successful; NULL if not

Closes the file currently associated with *stream* and reassigns *stream* to the file specified by *filename*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

frexp

double frexp(double _x_, int *_expptr_);

Include	MATH.H
x	Floating-point value
expptr	Pointer to stored integer exponent
Returns	The mantissa if successful; 0 if *x* is 0

Breaks down the floating-point value (x) into a mantissa (m) and an exponent (n) such that the absolute value of m is greater than or equal to 0.5 and less than 1.0, and $x = m * 2^n$.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

frexpl

long double frexpl(long double *x*, **int *** *expptr* **);**

Include	MATH.H
x	Long double-precision floating-point value
expptr	Pointer to stored integer exponent
Returns	The mantissa if successful; 0 if *x* is 0

Breaks down the long double-precision floating-point value (x) into a mantissa (m) and an exponent (n) such that the absolute value of m is greater than or equal to 0.5 and less than 1.0, and $x = m * 2^n$. Long double-precision floating-point version of **frexp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fscanf

int fscanf(FILE * *stream*, **const char *** *format* [[, *argument*]]... **);**

Include	STDIO.H
stream	Pointer to target stream
format	Format-control string
argument	Optional arguments
Returns	The number of fields that were successfully converted and assigned, or **EOF** for end-of-file

Reads data from the current position of *stream* into the locations given by *argument* (if any).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fseek

int fseek(FILE ****stream*, **long *offset*, **int** *origin*);**

Include	STDIO.H
stream	Pointer to target stream
offset	Number of bytes from origin
origin	Initial position: **SEEK_CUR, SEEK_END, SEEK_SET**
Returns	0 if successful; a nonzero value if not

Moves the file-position indicator associated with *stream* to a new location that is *offset* bytes from *origin*. It has limited use in text mode.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

fsetpos

int fsetpos(FILE ****stream*, **const fpos_t ****pos*);

Include	STDIO.H, ERRNO.H
stream	Pointer to target stream
pos	Position-indicator storage
Returns	0 if successful; a nonzero value if not
errno	**EBADF, EINVAL**

Sets the file-position indicator for *stream* to the value of *pos*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_fsopen

FILE *_fsopen(const char *_filename_, const char *_mode_, int _shflag_)

Include	STDIO.H, SHARE.H
filename	File path name
mode	Access permissions: **"r", "r+", "r+b", "r+t", "w", "w+", "w+b", "w+t", "a", "a+", "a+b", "a+t"**
shflag	**SH_COMPAT, SH_DENYRW, SH_DENYWR, SH_DENYRD, SH_DENYNO**
Returns	A pointer to the stream if successful; **NULL** if not

Opens a file as a stream and prepares it for shared reading and writing.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

fstat

int fstat(int _handle_, struct stat *_buffer_);

Include	SYS\TYPES.H, SYS\STAT.H, ERRNO.H
handle	Handle of open file
buffer	Pointer to structure to store results
Returns	0 if successful; –1 if not
errno	**EBADF**

Gets information about the open file associated with *handle* and stores it in the structure that *buffer* points to.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_fstrcat

char _far * _far _fstrcat(char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to destination string
string2	Far pointer to source string
Returns	A far pointer to the concatenated string

Appends *string2* to *string1*. Model-independent version of **strcat**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrchr

char _far * _far _fstrchr(const char _far *_string_, int _c_);

Include	STRING.H
string	Far pointer to source string
c	Character to be located
Returns	A far pointer to the first occurrence of *c* in *string*

Searches for the first occurrence of *c* in *string*. Model-independent version of **strchr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrcmp

int _far _fstrcmp(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1, *string2*	Far pointers to strings to compare
Returns	A value less than 0 if *string1* < *string2*; 0 if *string1* = *string2*; a value greater than 0 if *string1* > *string2*

Compares two strings. Model-independent version of **strcmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrcpy

char _far * _far _fstrcpy(char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to destination string
string2	Far pointer to source string
Returns	*string1*

Copies *string2* to *string1*. Model-independent version of **strcpy**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrcspn

size_t _far _fstrcspn(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to source string
string2	Far pointer to character set
Returns	The index of the first character in *string1* that belongs to the set of characters specified by *string2*

Finds first substring in *string1* consisting of characters not in *string2*. Model-independent version of **strcspn**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrdup

char _far * _far _fstrdup(const char _far *_string_);

Include	STRING.H
string	Far pointer to source string
Returns	A far pointer to the storage space containing the duplicate string

Duplicates *string*, copying it to memory allocated by **_fmalloc**. Model-independent version of **strdup**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstricmp

int _far _fstricmp(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1, *string2*	Far pointers to strings to compare
Returns	A value less than 0 if *string1* < *string2*; 0 if *string1* = *string2*; a value greater than 0 if *string1* > *string2*

Compares two strings without regard to case. Model-independent version of **stricmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrlen

size_t _fstrlen(const char _far *_string_);

Include	STRING.H
string	Far pointer to null-terminated string
Returns	The length in bytes of *string*, not including the terminating null character ('\0')

Finds length of string. Model-independent version of **strlen**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrlwr

char _far * _far _fstrlwr(char _far *_*string*);

Include	STRING.H
string	Far pointer to string to be converted
Returns	A far pointer to the converted string

Converts any uppercase letters in the given null-terminated *string* to lowercase. Other characters are not affected. Model-independent version of **strlwr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrncat

char _far * _far _fstrncat(char _far *_*string1*, const char _far *_*string2*, size_t *count*);

Include	STRING.H
string1	Far pointer to destination string
string2	Far pointer to source string
count	Number of characters appended
Returns	A far pointer to the concatenated string

Appends, at most, the first *count* characters of *string2* to *string1* and terminates the resulting string with a null character ('\0'). Model-independent version of **strncat**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrncmp

int _far _fstrncmp(const char _far *_string1_, const char _far *_string2_, size_t _count_);

Include	STRING.H
string1, *string2*	Far pointers to strings to compare
count	Number of characters compared
Returns	A value less than 0 if *string1* < *string2*; 0 if *string1* = *string2*; a value greater than 0 if *string1* > *string2*

Compares, at most, the first *count* characters of *string1* and *string2*. Model-independent version of **strncmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrncpy

char _far * _far _fstrncpy(char _far *_string1_, const char _far *_string2_, size_t _count_);

Include	STRING.H
string1	Far pointer to destination string
string2	Far pointer to source string
count	Number of characters copied
Returns	*string1*

Copies exactly *count* characters of *string2* to *string1*. Model-independent version of **strncpy**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrnicmp

int _far _fstrnicmp(const char _far *_string1_, const char _far *_string2_, size_t _count_);

Include	STRING.H
string1, *string2*	Far pointers to strings to compare
count	Number of characters compared
Returns	A value less than 0 if *string1* < *string2*; 0 if *string1* = *string2*; a value greater than 0 if *string1* > *string2*

Compares, at most, the first *count* characters of *string1* and *string2*, without regard to letter case. Model-independent version of **strnicmp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrnset

char _far * _far _fstrnset(char _far *_string_, int _c_, size_t _count_);

Include	STRING.H
string	Far pointer to string to be initialized
c	Character setting
count	Number of characters set
Returns	A far pointer to the altered string

Sets, at most, the first *count* characters of *string* to the character *c*. Model-independent version of **strnset**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrpbrk

char _far * _far _fstrpbrk(const char _far *_*string1*, const char _far *_*string2*);

Include	STRING.H
string1	Far pointer to source string
string2	Far pointer to character set
Returns	A far pointer to the first occurrence of any character from *string2* in *string1*

Finds the first occurrence of a character from one string in another. Model-independent version of **strpbrk**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrrchr

char _far * _far _fstrrchr(const char _far *_*string*, int *c*);

Include	STRING.H
string	Far pointer to searched string
c	Character to be located
Returns	A far pointer to the last occurrence of *c* in *string*

Finds the last occurrence of a given character in a string. Model-independent version of **strrchr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrrev

char _far * _far _fstrrev(char _far *_*string*_);

Include	STRING.H
string	Far pointer to string to be reversed
Returns	A far pointer to the altered string

Reverses the order of the characters in *string*. The terminating null character ('\0') remains in place. Model-independent version of **strrev**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrset

char _far * _far _fstrset(char _far *_*string*_, int _*c*_);

Include	STRING.H
string	Far pointer to string to be set
c	Far pointer to character setting
Returns	A far pointer to the altered string

Sets all of the characters of *string* to *c*, except the terminating null character ('\0'). Model-independent version of **strset**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrspn

size_t _far _fstrspn(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to searched string
string2	Far pointer to character set
Returns	An integer value specifying the length of the segment in *string1* consisting entirely of characters in *string2*

Returns the index of the first character in *string1* that does not belong to the set of characters specified by *string2*. This value is equivalent to the length of the initial substring of *string1* that consists entirely of characters from *string2*. Model-independent version of **strspn**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrstr

char _far * _far _fstrstr(const char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to searched string
string2	Far pointer to string to search for
Returns	A far pointer to the first occurrence of *string2* in *string1*

Finds the first occurrence of a given string in another string. Model-independent version of **strstr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrtok

char _far * _far _fstrtok(char _far *_string1_, const char _far *_string2_);

Include	STRING.H
string1	Far pointer to string containing token(s)
string2	Far pointer to set of delimiter characters
Returns	A far pointer to the first token in *string1*; successive calls against the same string return the subsequent tokens in the string

Reads *string1* as a series of zero or more tokens and *string2* as the set of characters serving as delimiters of the tokens in *string1*. Model-independent version of **strtok**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_fstrupr

char _far * _far _fstrupr(char _far *_string_);

Include	STRING.H
string	Far pointer to string to be capitalized
Returns	A far pointer to the converted string

Converts any lowercase letters in *string* to uppercase. Other characters are not affected. Model-independent version of **strupr**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

ftell

long ftell(FILE ****stream* **);

Include	STDIO.H, ERRNO.H
stream	Pointer to target stream
Returns	The current position if successful; –1L if not
errno	**EBADF, EINVAL**

Gets the current position of the file-position indicator associated with *stream*. Use with **fseek**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ftime

void ftime(struct timeb ****timeptr* **);

Include	SYS\TYPES.H, SYS\TIMEB.H
timeptr	Pointer to structure defined in SYS\TIMEB.H
Returns	No return value

Gets the current time and stores it in the structure that *timeptr* points to.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_fullpath

char *_fullpath(char *_buffer_, const char *_pathname_, size_t _maxlen_);

Include	STDLIB.H
buffer	Pointer to user path name buffer
pathname	Pointer to path name fragment
maxlen	Maximum length of *buffer*
Returns	A pointer to the absolute path in *buffer* if successful; **NULL** if not

Expands the path name fragment *pathname* to form a fully qualified path that is stored in *buffer*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

fwrite

size_t fwrite(const void *_buffer_, size_t _size_, size_t _count_, FILE *_stream_);

Include	STDIO.H
buffer	Pointer to data to be written
size	Item size in bytes
count	Maximum number of items to be written
stream	Pointer to target stream
Returns	The number of full items actually written, which may be less than *count* if an error occurs

Writes up to *count* items of length *size* from *buffer* to the output *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

gcvt

char *gcvt(double *value*, **int** *digits*, **char ***buffer* **);**

Include	STDLIB.H
value	Value to be converted
digits	Number of significant digits stored
buffer	Storage location for result
Returns	A pointer to the string of digits

Converts a floating-point *value* to a character string and stores the string in *buffer*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getactivepage

short _far _getactivepage(void);

Include	GRAPH.H
Returns	The number of the current active page

Determines the current active page.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_getarcinfo

short _far _getarcinfo(struct xycoord _far *_start_, struct xycoord _far *_end_, struct xycoord _far *_fillpoint_);

Include	GRAPH.H
start	Starting point of arc
end	Ending point of arc
fillpoint	Point to begin pie fill
Returns	A nonzero value if successful; 0 if not

Finds the endpoints in viewport coordinates of the arc most recently drawn by **_arc** or **_pie**.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_getbkcolor

long _far _getbkcolor(void);

Include	GRAPH.H
Returns	The current background color value

Determines the current background color.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

getc

int getc(FILE *_stream_);

Include	STDIO.H
stream	Pointer to target stream
Returns	The character read if successful; **EOF** to indicate an error or end-of-file

Reads a single character from the current *stream* position and increases the associated file-position indicator to point to the next character.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

getch

int getch(void);

Include	CONIO.H
Returns	The character read

Reads without echoing a single character from the console.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

getchar

int getchar(void);

Include	STDIO.H
Returns	The character read if successful; **EOF** to indicate an error or end-of-file

Reads a single character from the current standard input stream, **stdin**, and increases the associated file-position indicator to point to the next character.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

getche

int getche(void);

Include	CONIO.H
Returns	The character read

Reads a single character from the console and echoes the character read.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_getcolor

short _far _getcolor(void);

Include GRAPH.H

Returns The current color index

Determines the current color.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getcurrentposition

struct xycoord _far _getcurrentposition(void);

Include GRAPH.H

Returns The current position as an **xycoord** structure defined in GRAPH.H

Gets the current graphics output position in view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getcurrentposition_w

struct _wxycoord _far _getcurrentposition_w(void);

Include GRAPH.H

Returns The current position as a **_wxycoord** structure defined in GRAPH.H

Gets the current graphics output position in window coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

getcwd

char *getcwd(char *_buffer_, int _maxlen_);

Include	DIRECT.H, ERRNO.H
buffer	Storage location for path name
maxlen	Maximum length of path name
Returns	*buffer* if successful; **NULL** if not
errno	**ENOMEM, ERANGE**

Gets the full path name (including disk drive specification) of the current working directory and stores it at *buffer*.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getdcwd

char *_getdcwd(int _drive_, char *_buffer_, int _maxlen_);

Include	DIRECT.H, ERRNO.H
drive	Disk drive (A: = 1, B: = 2)
buffer	Storage location for path name
maxlen	Maximum length of path name
Returns	*buffer* if successful; **NULL** if not
errno	**ENOMEM, ERANGE**

Gets the full path name (including disk drive specification) of the current working directory on *drive* and stores it at *buffer*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_getdrive

int _getdrive(void);

Include	DIRECT.H
Returns	The integer indicating the current working drive (A: = 1, B: = 2)

Gets the current working drive.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

getenv

char *getenv(const char *_varname_);

Include	STDLIB.H
varname	Name of environment variable
Returns	A pointer to the environment table entry containing the current string value of *varname*; **NULL** if the variable is not defined

Searches the list of environment variables for an entry corresponding to *varname*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getfillmask

unsigned char _far * _far _getfillmask(unsigned char _far *_mask_);

Include	GRAPH.H
mask	Mask array
Returns	**NULL** if no mask is set

Returns the current fill mask used in the **_floodfill**, **_pie**, **_ellipse**, **_polygon**, and **_rectangle** functions.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getfontinfo

short _far _getfontinfo(struct _fontinfo _far *_fontbuffer_);

Include	GRAPH.H
fontbuffer	Font information
Returns	–1 if a font has not been registered or loaded

Returns the current font characteristics in a **_fontinfo** structure defined in GRAPH.H.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getgtextextent

short _far _getgtextextent(unsigned char _far *_text_);

Include	GRAPH.H
text	Text to be analyzed
Returns	The width in pixels; –1 if a font has not been registered

Returns the width required to print the text string in the current font, using **_outgtext**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getgtextvector

short xycoord _far _getgtextvector(void);

Include	GRAPH.H
Returns	Vector determining the direction of font text rotation as an **xycoord** structure defined in GRAPH.H

Finds the orientation for font text output by **_setgtextvector**.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getimage

void _far _getimage(short *x1*, **short** *y1*, **short** *x2*, **short** *y2*, **char _huge *** *image* **);**

Include	GRAPH.H
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
image	Storage buffer for screen image
Returns	No return value; check **_grstatus**

Stores the screen image defined by a specified bounding rectangle into the buffer *image*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getimage_w

void _far _getimage_w(double *wx1*, **double** *wy1*, **double** *wx2*, **double** *wy2*,
char _huge * *image* **);**

Include	GRAPH.H
wx1, *wy1*	Upper left corner of bounding rectangle
wx2, *wy2*	Lower right corner of bounding rectangle
image	Storage buffer for screen image
Returns	No return value; check **_grstatus**

Stores the screen image defined by a specified bounding rectangle into the buffer *image*. Defines the bounding rectangle with the window coordinates (*wx1*, *wy1*) and (*wx2*, *wy2*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getimage_wxy

void _far _getimage_wxy(struct _wxycoord _far *pwxy1,
struct _wxycoord _far *pwxy2, char _huge *image);

Include	GRAPH.H
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
image	Storage buffer for screen image
Returns	No return value; check **_grstatus**

Stores the screen image defined by a specified bounding rectangle into the buffer *image*. Defines the bounding rectangle with the window coordinate pairs *pwxy1* and *pwxy2*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getlinestyle

unsigned short _far _getlinestyle(void);

Include	GRAPH.H
Returns	The default mask if no mask has been set

Returns the current line-style mask, which is used by the **_lineto**, **_polygon**, and **_rectangle** functions.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getphyscoord

struct xycoord _far _getphyscoord(short *x*, short *y*);

Include	GRAPH.H
x, *y*	View coordinate to translate
Returns	The physical coordinates in an **xycoord** structure defined in GRAPH.H

Translates the view coordinate (*x*, *y*) to physical coordinates, returning them in an **xycoord** structure.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

getpid

int getpid(void);

Include	PROCESS.H
Returns	The process identification

Gets the identification number of the calling process.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getpixel

short _far _getpixel(short *x*, short *y*);

Include	GRAPH.H
x, *y*	Pixel location
Returns	The color index if successful; –1 if not

Determines the pixel value at the location specified in viewport coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getpixel_w

short _far _getpixel_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	Pixel location
Returns	The color index if successful; –1 if not

Determines the pixel value at the location specified in window coordinates.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

gets

char *gets(char ***buffer* **);

Include	STDIO.H
buffer	Storage location for input string
Returns	The argument if successful; **NULL** if not, or if end-of-file is reached

Reads a line from the standard input stream, **stdin**, and stores it at *buffer*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_gettextcolor

short _far _gettextcolor(void);

Include	GRAPH.H
Returns	Index of the current text color

Determines the current text attribute.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_gettextcursor

short _far _gettextcursor(void);

Include	GRAPH.H
Returns	The current cursor attribute if successful; –1 if not

Determines the current cursor shape in text modes.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_gettextposition

struct rccoord _far _gettextposition(void);

Include	GRAPH.H
Returns	The text position as an **rccoord** structure defined in GRAPH.H

Gets the current text position as an **rccoord** structure.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_gettextwindow

void _far _gettextwindow(short _far ****r1*, short _far ****c1***, short _far ****r2***,**
short _far ****c2* **);

Include	GRAPH.H
r1	Top row of current text window
c1	Leftmost column of current text window
r2	Bottom row of current text window
c2	Rightmost column of current text window
Returns	No return value

Gets the boundaries of the current text window.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_getvideoconfig

struct videoconfig _far * _far _getvideoconfig(struct videoconfig _far *_*config*);

Include	GRAPH.H
config	Configuration information
Returns	The video configuration information in a **videoconfig** structure defined in GRAPH.H

Gets information about the current graphics environment as a **videoconfig** structure.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_getviewcoord

struct xycoord _far _getviewcoord(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	Physical coordinates to translate
Returns	View coordinates in an **xycoord** structure defined in GRAPH.H

Translates the specified physical coordinates to view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getviewcoord_w

struct xycoord _far _getviewcoord_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	Window coordinates to translate
Returns	View coordinates in an **xycoord** structure defined in GRAPH.H

Translates the specified window coordinates to view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getviewcoord_wxy

struct xycoord _far _getviewcoord_wxy(struct _wxycoord _far * *pwxyl* **);**

Include	GRAPH.H
pwxyl	Window coordinate to translate
Returns	View coordinates in an **xycoord** structure defined in GRAPH.H

Translates the specified window coordinate structure to view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getvisualpage

short _far _getvisualpage(void);

Include	GRAPH.H
Returns	The number of the current visual page

Gets the current visual page number.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

getw

int getw(FILE ***stream* **);

Include	STDIO.H
stream	Pointer to target stream
Returns	The integer value read if successful; **EOF** to indicate failure or end-of-file

Reads the next binary value of type **int** from *stream* and increases the associated file pointer to point to the next unread character.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_getwindowcoord

struct _wxycoord _far _getwindowcoord(short *x*, short *y*);

Include	GRAPH.H
x, *y*	Physical coordinates to translate
Returns	The coordinates in an **_wxycoord** structure defined in GRAPH.H

Translates the view coordinates (*x*, *y*) to window coordinates and returns them in an **_wxycoord** structure.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_getwritemode

short _far _getwritemode(void);

Include	GRAPH.H
Returns	**_GPSET**, **_GXOR**, **_GAND**, **_GOR**, **_GPRESET**; –1 if an error occurs

Returns the current logical write mode used by **_lineto**, **_rectangle**, and **_polygon**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

gmtime

struct tm *gmtime(const time_t **timer*);

Include	TIME.H
timer	Pointer to stored Greenwich mean time value
Returns	A pointer to the structure result

Converts the *timer* value to a structure.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_grstatus

short _far _grstatus(void);

Include	GRAPH.H
Returns	The status of the most recently called graphics function

Determines whether errors or warnings were generated by the most recently called graphics function.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

halloc

void _huge *halloc(long *num*, **size_t** *size* **);**

Include	MALLOC.H
num	Number of elements
size	Length in bytes of each element
Returns	A pointer to the allocated space

Allocates a huge array from the operating system consisting of *num* elements, each of which is *size* bytes long.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_harderr

void _harderr(void(_far *_handler_)( ));

Include	DOS.H
handler	New INT 0x24 handler
Returns	No return value

Establishes the user-defined routine that *handler* points to as the handler for DOS INT 0x24, which is invoked when a hardware error occurs during the execution of an I/O request.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_hardresume

void _hardresume(int *result* **);**

Include	DOS.H
result	**_HARDERR_IGNORE, _HARDERR_RETRY, _HARDERR_ABORT, _HARDERR_FAIL**
Returns	No return value

Allows a user-defined hard I/O error handler to return to DOS.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_hardretn

void _hardretn(int *error* **);**

Include	DOS.H
error	Error to return from
Returns	No return value

Allows a user-defined hard I/O error handler to return directly to the application program.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_heapadd

int _heapadd(void _far ****memblock*, **size_t *size* **);**

Include	MALLOC.H
memblock	Far pointer to heap memory
size	Size in bytes of memory to add
Returns	0 if successful; –1 if not

Adds an unused piece of memory to the heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_heapchk

int _heapchk(void);

Include	MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks the heap for minimal consistency.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_heapmin

int _heapmin(void);

Include	MALLOC.H
Returns	0 if successful; –1 if not

Releases unused memory in the heap back to the operating system.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_heapset

int _heapset(unsigned int *fill* **);**

Include	MALLOC.H
fill	Fill character
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks the heap for minimal consistency and then sets the heap's free entries with the *fill* value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_heapwalk

int _heapwalk(_HEAPINFO **entryinfo* **);**

Include	MALLOC.H
entryinfo	Structure to contain information about the next heap entry, defined in MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADPTR, _HEAPBADBEGIN, _HEAPBADNODE, _HEAPEND**

Walks through the heap, one entry per call, returning a pointer to a **_HEAPINFO** structure that contains information about the next heap entry.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

hfree

void hfree(void _huge **memblock* **);**

Include	MALLOC.H
memblock	Pointer to allocated memory block
Returns	No return value

Deallocates a huge memory block obtained with **halloc** and returns the freed memory to the operating system.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

hypot

double hypot(double *x*, double *y*);

Include	MATH.H, ERRNO.H
x, *y*	Floating-point values
Returns	The length of the hypotenuse if successful; **HUGE_VAL** on overflow
errno	**ERANGE**

Calculates the length of the hypotenuse of a right triangle, given the length of the two sides *x* and *y*.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

hypotl

long double hypotl(long double *x*, long double *y*);

Include	MATH.H, ERRNO.H
x, *y*	Long double-precision floating-point values
Returns	The length of the hypotenuse if successful; **_LHUGE_VAL** on overflow
errno	**ERANGE**

Calculates the length of the hypotenuse of a right triangle, given the length of the two sides *x* and *y*, using the long double-precision floating-point version of arguments and return values.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_imagesize

long _far _imagesize(short *x1*, **short** *y1*, **short** *x2*, **short** *y2* **);**

Include	GRAPH.H
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
Returns	The storage size of the image in bytes

Calculates the number of bytes needed to store the image defined by the bounding rectangle specified by the coordinates *(x1, y1)* and *(x2, y2)*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_imagesize_w

long _far _imagesize_w(double *wx1*, **double** *wy1*, **double** *wx2*, **double** *wy2* **);**

Include	GRAPH.H
wx1, *wy1*	Upper left corner of bounding rectangle
wx2, *wy2*	Lower right corner of bounding rectangle
Returns	The storage size of the image in bytes

Calculates the number of bytes needed to store the image defined by the bounding rectangle specified by the window coordinates (*wx1, wy1*) and (*wx2, wy2*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_imagesize_wxy

long _far _imagesize_wxy(struct _wxycoord _ far ***_pwxy1_,**
struct _wxycoord _far ***_pwxy2_ **);

Include	GRAPH.H
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
Returns	The storage size of the image in bytes

Calculates the number of bytes needed to store the image defined by the bounding rectangle specified by the window coordinate pairs *pwxy1* and *pwxy2*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

inp

int inp(unsigned *port* **);**

Include	CONIO.H
port	Port number
Returns	The byte read from *port*

Reads a byte from the specified port. (Note: For protected-mode libraries, use a .DEF file containing this line: SEGMENTS _IOSEG CLASS "IOSEG_CODE" IOPL.)

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

inpw

unsigned inpw(unsigned *port* **);**

Include	CONIO.H
port	Port number
Returns	The word read from *port*

Reads a word from the specified port. (Note: For protected-mode libraries, use a .DEF file containing this line: SEGMENTS _IOSEG CLASS "IOSEG_CODE" IOPL.)

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

intdos

int intdos(union REGS ***_inregs_, union REGS ***_outregs_ **);**

Include	DOS.H
inregs	Register values on call
outregs	Register values on return
Returns	The value of the AX register after the system call is completed

Invokes the DOS system call specified by register values defmed in *inregs* and returns the effect of the system call in *outregs*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

intdosx

int intdosx(union REGS ****inregs*, **union REGS ****outregs*, **struct SREGS ****segregs* **);

Include	DOS.H
inregs	Register values on call and return
outregs	Register values on return
segregs	Segment-register values on call and on return
Returns	The value of the AX register after the system call is completed

Invokes the DOS system call specified by register values defined in *inregs* and *segregs* and returns the effect of the system call in *outregs* and *segregs*. It accepts segment-register values specifying segments and pointers.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

int86

int int86(int *intnum*, **union REGS ****inregs*, **union REGS ****outregs* **);**

Include	DOS.H
intnum	Interrupt number
inregs	Register values on call
outregs	Register values on return
Returns	The value of the AX register after the interrupt returns

Executes the 8086-processor-family interrupt specified by the interrupt number *intnum*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

int86x

int int86x(int *intnum*, **union REGS ***_inregs_, **union REGS ***_outregs_,
struct SREGS ***_segregs_ **);

Include	DOS.H
intnum	Interrupt number
inregs	Register values on call
outregs	Register values on return
segregs	Segment-register values on call and on return
Returns	The value of the AX register after the interrupt returns

Executes the 8086-processor-family interrupt specified by the interrupt number *intnum*. It accepts segment-register values specifying segments and pointers.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

isalnum

int isalnum(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for alphanumeric character ('A'–'Z', 'a'–'z', or '0'–'9').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isalpha

int isalpha(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for letter ('A'–'Z' or 'a'–'z').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isascii

int isascii(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for ASCII character (0x00 – 0x7f).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isatty

int isatty(int *handle* **);**

Include	IO.H
handle	Handle referring to device to be tested
Returns	A nonzero value if the device is a character device; 0 if not

Determines whether *handle* is associated with a character device (a terminal, console, printer, or serial port).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

iscntrl

int iscntrl(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for control character (0x00 – 0x 1f or 0x7f).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isdigit

int isdigit(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for digit ('0'–'9').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

Isgraph

int isgraph(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for graphical characters (any printable character except white space).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

islower

int islower(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for lowercase character ('a'–'z').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isprint

int isprint(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for printable character (0x20 – 0x7e).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ispunct

int ispunct(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for punctuation character.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isspace

int isspace(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for white space character (0x09 – 0x0d or 0x20).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isupper

int isupper(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for uppercase character ('A'–'Z').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

isxdigit

int isxdigit(int *c* **);**

Include	CTYPE.H
c	Integer to be tested
Returns	A nonzero value if the integer satisfies the test condition; 0 if not

Tests for hexadecimal digit ('A'–'F','a'–'f', or '0'–'9').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

itoa

char *itoa(int *value*, **char ***string*, **int** *radix* **);**

Include	STDLIB.H
value	Number to be converted
string	String result
radix	Base of *value*
Returns	A pointer to *string*

Converts the digits of the given *value* to a null-terminated character string and stores the result (up to 17 bytes) in *string*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

jn

double jn(int *n*, **double** *x* **);**

Include	MATH.H
n	Integer order
x	Floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order *n*).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_jnl

long double _jnl(int *n*, long double *x*);

Include	MATH.H
n	Integer order
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order *n*). Long double-precision floating-point version of **jn**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

j1

double j1(double *x*);

Include	MATH.H
x	Floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order 1).

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_j1l

long double _j1l(long double *x*);

Include	MATH.H
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order 1). Long double-precision floating-point version of **j1**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

j0

double j0(double *x* **);**

Include	MATH.H
x	Floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order 0).

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_j0l

long double _j0l(long double *x* **);**

Include	MATH.H
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*

Returns the Bessel function of the first kind (order 0). Long double-precision floating-point version of **j0**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

kbhit

int kbhit(void);

Include	CONIO.H
Returns	A nonzero value if a key has been pressed; 0 if not

Checks the console for an unread keystroke.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

labs

long labs(long *n*);

Include	STDLIB.H or MATH.H
n	Long integer value
Returns	The absolute value of *n*

Calculates the long-integer absolute value of *n*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

ldexp

double ldexp(double *x*, int *exp*);

Include	MATH.H, ERRNO.H
x	Floating-point value
exp	Integer exponent
Returns	The value of $x * 2^{exp}$ if successful; ± **HUGE_VAL** (depending on the sign of *x*) on overflow
errno	**ERANGE**

Converts the mantissa and exponent to a floating-point value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ldexpl

long double ldexpl(long double *x*, **int** *exp* **);**

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
exp	Integer exponent
Returns	The value of $x * 2^{exp}$ if successful; ±**_LHUGE_VAL** (depending on the sign of *x*) on overflow
errno	**ERANGE**

Converts the mantissa and exponent to a floating-point value. Long double-precision floating-point version of **ldexp**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

ldiv

ldiv_t ldiv(long int *numer*, **long int** *denom* **);**

Include	STDLIB.H
numer	Numerator
denom	Denominator
Returns	The quotient and remainder as an **ldiv_t** structure

Computes the quotient and remainder for a pair of long integer values.

■ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

lfind

void *lfind(const void *_key_, const void *_base_, unsigned int *_num_, unsigned int _width_, int(*_compare_)(const void *_elem1_, const void *_elem2_));

Include	SEARCH.H
key	Object to search for
base	Pointer to base of search data
num	Number of elements
width	Element width
compare	Pointer to comparison function
elem1	Pointer to the key for the search
elem2	Pointer to the array element to be compared with the key
Returns	A pointer to an array element corresponding to *key* if successful; **NULL** if not

Performs a linear search for the value *key* in an array of *num* elements, each of *width* bytes in size.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_lineto

short _far _lineto(short _x_, short _y_);

Include	GRAPH.H
x, *y*	End point
Returns	A nonzero value if successful; 0 if not

Draws a line from the current position up to and including the view coordinate (*x*, *y*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_lineto_w

short _far _lineto_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	End point
Returns	A nonzero value if successful; 0 if not

Draws a line from the current position up to and including the window coordinate (*wx*, *wy*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

localeconv

struct lconv *localeconv(void);

Include	LOCALE.H
Returns	A pointer to a structure of type **lconv**

Gets detailed information on locale settings.

■ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

localtime

struct tm *localtime(const time_t ***timer* **);

Include	TIME.H
timer	Pointer to stored time structure
Returns	Pointer to a structure resolved; **NULL** if time cannot be interpreted

Converts a time stored as a **long** value to a structure of type **tm**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

locking

int locking(int *handle*, **int** *mode*, **long** *nbytes* **);**

Include	SYS\LOCKING.H, IO.H, ERRNO.H
handle	File handle
mode	**LK_LOCK, LK_RLCK, LK_NBLCK, LK_NBRLCK, LK_UNLCK**
nbytes	Number of bytes to lock
Returns	0 if successful; –1 if not
errno	**EACCESS, EBADF, EDEADLOCK, EINVAL**

Locks or unlocks *nbytes* bytes of the file specified by *handle*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

log

double log(double *x* **);**

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	The natural logarithm of *x*
errno	**EDOM, ERANGE**

Calculates the natural logarithm of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

logl

long double logl(long double *x*);

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	The natural logarithm of *x*
errno	**EDOM, ERANGE**

Calculates the natural logarithm of *x*. Long double-precision floating-point version of **log**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

log10

double log10(double *x*);

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	The base-10 logarithm of *x*
errno	**EDOM, ERANGE**

Calculates the base-10 logarithm of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

log10l

long double log10l(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	The base-10 logarithm of *x*
errno	**EDOM, ERANGE**

Calculates the base-10 logarithm of *x*. Long double-precision floating-point version of **log10**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

longjmp

void longjmp(jmp_buf *env*, **int** *value* **);**

Include	SETJMP.H
env	Variable in which environment is stored
value	Value to be returned to **setjmp** call
Returns	No return value

Restores a stack environment and execution locale previously saved in *env* by **setjmp**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_lrotl

unsigned long _lrotl(unsigned long *value*, int *shift*);

Include	STDLIB.H
value	Value to be rotated
shift	Number of bits to shift
Returns	The rotated value

Rotates the *value* to the left by *shift* bits.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_lrotr

unsigned long _lrotr(unsigned long *value*, int *shift*);

Include	STDLIB.H
value	Value to be rotated
shift	Number of bits to shift
Returns	The rotated value

Rotates the *value* to the right by *shift* bits.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

lsearch

void *lsearch(const void *_key_, const void *_base_, unsigned int *_num_, unsigned int _width_, int(*_compare_)(const void *_elem1_, const void *_elem2_));

Include	SEARCH.H
key	Object to search for
base	Pointer to base of search data
num	Number of elements
width	Element width
compare	Pointer to comparison function
elem1	Pointer to the key for the search
elem2	Pointer to the array element to be compared with the key
Returns	A pointer to the array element *base* that matches *key* if the key is found; a pointer to the newly added item at the end of the array otherwise

Performs a linear search for the value *key* in an array of *num* elements, each of *width* bytes in size. If the value is not found, it adds *key* to the end of the list.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

lseek

long lseek(int _handle_, long _offset_, int _origin_);

Include	IO.H, STDIO.H, ERRNO.H
handle	Handle referring to open file
offset	Number of bytes from *origin*
origin	**SEEK_SET, SEEK_CUR, SEEK_END**
Returns	The offset in bytes of the new position if successful; –1L if not
errno	**EBADF, EINVAL**

Moves the file pointer associated with *handle* to a new location that is *offset* bytes from *origin*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ltoa

char *ltoa(long *value*, **char ***string*, **int** *radix* **);**

Include	STDLIB.H
value	Number to be converted
string	String result
radix	Base of *value*
Returns	A pointer to *string*

Converts the digits of *value* to a null-terminated character string and stores the result (up to 33 bytes) in *string*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_makepath

void _makepath(char ***path*, **char ***drive*, **char ***dir*, **char ***fname*, **char ***ext* **);

Include	STDLIB.H
path	Full path-name buffer
drive	Drive letter
dir	Directory path
fname	File name
ext	File extension
Returns	No return value

Creates a single path name, composed of a drive letter, directory path, file name, and file-name extension.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

malloc

void *malloc(size_t *size* **);**

Include	STDLIB.H or MALLOC.H
size	Number of bytes to allocate
Returns	A pointer to the allocated space

Allocates a memory block of at least *size* bytes.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

matherr

int matherr(struct exception ***except* **);

Include	MATH.H
except	Math exception information
Returns	A nonzero value if successful; 0 if not

Processes errors generated by the functions of the math library. The user can provide a different **matherr** definition for special treatment of math errors.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_matherrl

int _matherrl(struct _exceptionl ***except* **);

Include	MATH.H
except	Math exception information
Returns	A nonzero value if successful; 0 if not

Processes errors generated by the long double-precision functions of the math library.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

max

type **max(** *type a*, *type b* **);**

Include	STDLIB.H
type	Any numerical data type
a, *b*	Values to compare
Returns	The larger of the two arguments

Compares two values of the same data type, returning the larger one.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_memavl

size_t _memavl(void);

Include	MALLOC.H
Returns	The size in bytes as an unsigned integer

Returns the approximate amount of memory available for dynamic memory allocation in the near heap (default data segment).

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

memccpy

void *memccpy(void *_dest_, void *_src_, int _c_, unsigned int _count_);

Include	STRING.H or MEMORY.H
dest	Pointer to destination
src	Pointer to source
c	Last character to copy
count	Number of characters
Returns	A pointer to the byte in *dest* that immediately follows the copied character (*c*) if successful; **NULL** if *count* bytes copied before *c* was found

Copies zero or more bytes of the source to the destination, up to and including the first occurrence of the character *c* or until *count* bytes have been copied, whichever comes first.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

memchr

void *memchr(const void *_buf_, int _c_, size_t _count_);

Include	STRING.H or MEMORY.H
buf	Pointer to buffer
c	Character to search for
count	Maximum number of characters to scan
Returns	A pointer to the first location of *c* in *buf* if successful; **NULL** if not

Looks for the first occurrence of a character in the first *count* bytes of the buffer.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

memcmp

int memcmp(const void ***_buf1_, const void ***_buf2_**, size_t** _count_ **);**

Include	STRING.H or MEMORY.H
buf1, *buf2*	Pointers to buffers to compare
count	Maximum number of characters to compare
Returns	A value less than, equal to, or greater than 0 if the size of the buffer pointed to by *buf1* is less than, equal to, or greater than the size of the buffer pointed to by *buf2*.

Compares the first *count* bytes of two buffers and returns a value indicating their relationship.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

memcpy

void *memcpy(void ***_dest_, const void ***_src_**, size_t** _count_ **);**

Include	STRING.H or MEMORY.H
dest	Pointer to destination buffer
src	Pointer to the buffer to copy from
count	Number of characters to copy
Returns	*dest*

Copies *count* bytes of *src* to *dest*. The first *count* bytes of *src* and *dest* should not overlap.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

memicmp

int memicmp(void ****buf1*, **void ****buf2*, **unsigned int *count* **);**

Include	STRING.H or MEMORY.H
buf1, *buf2*	Pointers to buffers to compare
count	Number of characters
Returns	A value less than, equal to, or greater than 0 if the size of the buffer pointed to by *buf1* is less than, equal to, or greater than the size of the buffer pointed to by *buf2*.

Compares the first *count* bytes of *buf1* and *buf2* byte-by-byte, without regarding case (all letters are considered lowercase), and returns a value indicating their relationship.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_memmax

size_t _memmax(void);

Include	MALLOC.H
Returns	The size of the largest contiguous block of memory that can be allocated from the near heap if successful; 0 if nothing more can be allocated from the near heap

Determines the size of the largest contiguous free memory block in the near heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

memmove

void *memmove(void *_dest_, const void *_src_, size_t _count_);

Include	STRING.H
dest	Pointer to destination buffer
src	Pointer to source buffer
count	Number of characters to copy
Returns	The value of *dest*

Copies *count* characters from *src* to *dest* and handles overlap properly.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

memset

void *memset(void *_dest_, int _c_, size_t _count_);

Include	STRING.H or MEMORY.H
dest	Pointer to destination buffer
c	Character to set to
count	Number of characters to be set
Returns	A pointer to *dest*

Sets the first *count* bytes of *dest* to the character *c*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

min

type **min(** *type a, type b* **);**

Include	STDLIB.H
type	Any numeric data type
a, b	Values to compare
Returns	The smaller of the two arguments

Compares two values of the same data type and returns the smaller of them.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

mkdir

int mkdir(char ***_dirname_ **);

Include	DIRECT.H, ERRNO.H
dirname	Path name for new directory
Returns	0 if successful; –1 if not
errno	**EACCESS, ENOENT**

Creates a new directory with the specified directory name.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

mktemp

char *mktemp(char ***_template_ **);

Include	IO.H
template	File-name pattern
Returns	A pointer to the modified template if successful; **NULL** if not

Creates a unique file name by modifying the given file-name pattern.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

mktime

time_t mktime(struct tm ****timeptr* **);

Include	TIME.H
timeptr	Local time structure
Returns	The calendar time encoded as a type **time_t** value if successful; –1 cast as type **time_t** if not

Converts the local time to a calendar value.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

modf

double modf(double *x*, **double ****intptr* **);**

Include	MATH.H
x	Floating-point value
intptr	Pointer to stored integer portion
Returns	The signed fractional portion of *x*

Breaks down the floating-point value *x* into fractional and integer parts with the same sign as *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

modfl

long double modfl(long double *x*, long double **intptr*);

Include	MATH.H
x	Long double-precision floating-point value
intptr	Pointer to stored integer portion
Returns	The signed fractional portion of *x*

Breaks down the long double-precision floating-point value of *x* into fractional and integer parts with the same sign as *x*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

movedata

void movedata(unsigned int *srcseg*, **unsigned int** *srcoff*, **unsigned int** *destseg*, **unsigned int** *destoff*, **unsigned int** *count* **);**

Include	STRING.H or MEMORY.H
srcseg	Segment address of source
srcoff	Segment offset of source
destseg	Segment address of destination
destoff	Segment offset of destination
count	Number of bytes
Returns	No return value

Copies *count* bytes from the source address specified by *srcseg*:*srcoff* to the destination address specified by *destseg*:*destoff*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_moveto

struct xycoord _far _moveto(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	Target position
Returns	The coordinates of the previous position in an **xycoord** structure

Moves the current graphics-output position to the specified view coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_moveto_w

struct _wxycoord _far _moveto_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	Target position
Returns	The coordinates of the previous position in a **_wxycoord** structure

Moves the current graphics-output position to the specified window coordinates.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_msize

size_t _msize(void ***_memblock_ **);

Include	MALLOC.H
memblock	Pointer to memory block
Returns	The size in bytes

An unsigned integer representing the size in bytes of the memory block allocated by a call to **calloc**, **malloc**, or **realloc**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_ncalloc

void _near *_ncalloc(size_t *num*, size_t *size*);

Include	MALLOC.H
num	Number of elements
size	Length in bytes of each element
Returns	A near pointer to the allocated space if successful; **NULL** if there is insufficient memory available

Allocates storage for an array of *num* elements, each of length *size* bytes in the near heap. Storage is initialized to zero.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_nexpand

void _near *_nexpand(void _near **memblock*, size_t *size*);

Include	MALLOC.H
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A near pointer to the resized block if successful; **NULL** if there is insufficient memory

Changes the size of a previously allocated memory block by attempting to expand or contract the block without moving its location in the near heap.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_nfree

void _nfree(void _near *_memblock_);

Include	MALLOC.H
memblock	Allocated memory block in the near heap
Returns	No return value

Frees a memory block in the near heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nheapchk

int _nheapchk(void);

Include	MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks for minimal consistency on the dynamically allocated memory space in the default data segment, or "near heap." The consistency check determines whether all the near heap entries are within the bounds of the heap's current memory allocation.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nheapmin

int _nheapmin(void);

Include	MALLOC.H
Returns	0 if successful; –1 if not

Releases unused memory in the near heap back to the operating system.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nheapset

int _nheapset(unsigned int *fill* **);**

Include	MALLOC.H
fill	Fill character
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADBEGIN, _HEAPBADNODE**

Checks for minimal consistency in the near heap and then sets the heap's free entries with the *fill* value.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nheapwalk

int _nheapwalk(_HEAPINFO * *entryinfo* **);**

Include	MALLOC.H
entryinfo	Structure to contain information about the next near heap entry defined in MALLOC.H
Returns	**_HEAPOK, _HEAPEMPTY, _HEAPBADPTR, _HEAPBADBEGIN, _HEAPBADNODE, _HEAPEND**

Walks through the near heap, one entry per call, returning a pointer to a **_HEAPINFO** structure that contains information about the next near heap entry.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nmalloc

void _near *_nmalloc(size_t *size* **);**

Include	MALLOC.H
size	Number of bytes to allocate
Returns	A near pointer to the allocated space if successful; **NULL** if not

Allocates a memory block of at least *size* bytes in the near heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nmsize

size_t _nmsize(void _near *_memblock_);

Include	MALLOC.H
memblock	Pointer to near memory block
Returns	The size in bytes

An unsigned integer representing the size in bytes of the near-heap memory block allocated by a call to **_ncalloc**, **_nmalloc**, or **_nrealloc**

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nrealloc

void _near *_nrealloc(void _near *_memblock_, size_t _size_);

Include	MALLOC.H
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A near pointer to the reallocated memory block if successful; **NULL** if not

Changes the size and possibly the location of a memory block in the near heap.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_nstrdup

char _near * _far _nstrdup(const char _far *_string_);

Include	STRING.H
string	Source string
Returns	A near pointer to the storage space containing the duplicate string

Duplicates *string*, copying it to memory allocated by **_nmalloc**. Model-independent form of **strdup**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

onexit

onexit_t onexit(onexit_t *func* **);**

Include	STDLIB.H
func	Function to call on exit
Returns	A pointer to the function if successful; NULL if there is no space left to store the function pointer

Creates a register of functions to be called (last-in, first-out) when the program terminates normally.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

open

int open(char ***_filename_, int** *oflag* [[**, int** *pmode*]] **);**

Include	SYS\TYPES.H, SYS\STAT.H, IO.H, ERRNO.H, FCNTL.H
filename	File path name
oflag	**O_APPEND, O_BINARY, O_CREAT, O_EXCL, O_RDONLY, O_RDWR, O_TEXT, O_TRUNC, O_WRONLY**
pmode	**S_IWRITE, S_IREAD, S_IREAD | S_IWRITE**
Returns	A handle for the opened file if successful; –1 if not
errno	**EACCESS, EEXIST, EINVAL, EMFILE, ENOENT**

Opens the file specified by *filename* and prepares the file for subsequent reading or writing, as defined by *oflag* and, optionally, *pmode*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_outgtext

void _far _outgtext(unsigned char _far *text);

Include	GRAPH.H
text	Text to be output
Return	No return value

Outputs the null-terminated string *text* using the current font and rotation on the screen at the current graphics-output position.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_outmem

void _far _outmem(unsigned char _far *text, short length);

Include	GRAPH.H
text	Text to be output
length	Number of characters to be output
Returns	No return value

Outputs the specified number of characters from a buffer.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

outp

int outp(unsigned port, int databyte);

Include	CONIO.H
port	Port number
databyte	Output value
Returns	The output data

Writes a byte to the specified output port. (Note: For protected-mode libraries, use a .DEF file containing this line: SEGMENTS _IOSEG CLASS, "IOSEG_CODE" IOPL.)

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

outpw

unsigned outpw(unsigned *port*, **unsigned** *dataword* **);**

Include	CONIO.H
port	Port number
dataword	Output value
Returns	The output data

Writes a word to the specified output port. (Note: For protected-mode libraries, use a .DEF file containing this line: SEGMENTS _IOSEG CLASS "IOSEG_CODE" IOPL.)

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_outtext

void _far _outtext(unsigned char _far ***_text_ **);

Include	GRAPH.H
text	Text to be output
Return	No return value

Outputs the null-terminated string, *text*, at the current text position using the current text color.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_pclose

int _pclose(FILE ***_stream_ **);

Include	STDIO.H
stream	Return value from previous call to **_popen**
Returns	The exit status of the child command if successful; -1 if not

Closes *stream* and waits for the associated child command.

□ ANSI □ DOS ■ OS/2 ■ UNIX ■ XENIX

perror

void perror(const char ***_string_);**

Include	STDIO.H
string	User-supplied message
Returns	No return value

Prints an error message to **stderr**. The *string* argument is printed first, followed by a colon, then by the system error message for the last library call that produced the error, and finally by a newline character.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_pg_analyzechart

short _far _pg_analyzechart(chartenv _far ***_env_, char _far * _far ***_categories_**, float _far ***_values_**, short** _n_ **);**

Include	PGCHART.H
env	Chart environment variable
categories	Array of category variables
values	Array of data values
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Analyzes a single series of data and fills the chart environment with default values for a single-series bar, column, or line chart, depending on the type specified in the function call.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_pg_analyzechartms

short _far _pg_analyzechartms(chartenv _far ****env*, char _far * _far ****categories***,**
float _far ****values*, short** *nseries***, short** *n***, short** *arraydim***,**
char _far * _far ****serieslabels*);**

Include	PGCHART.H
env	Chart environment variable
categories	Array of category variables
values	Array of data values
nseries	Number of series to chart
n	Number of data values to chart
arraydim	Row dimension of data array
serieslabels	Array of labels for series
Returns	0 if successful; a nonzero value if not

Analyzes a multiple series of data and fills the chart environment with default values for a multiseries bar, column, or line chart, depending on which type is specified in the function call.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_analyzepie

short _far _pg_analyzepie(chartenv _far ****env*, char _far * _far ****categories***, float _far ****values***, short _far ****explode***, short** *n* **);**

Include	PGCHART.H
env	Chart environment structure
categories	Array of category labels
values	Array of data values
explode	Array of explode flags
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Analyzes a single series of data and fills the chart environment for a pie chart using the data contained in the array *values*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_analyzescatter

short _far _pg_analyzescatter(chartenv _far ****env*, float _far ****xvalues***, float _far ****yvalues***, short** *n* **);**

Include	PGCHART.H
env	Chart environment structure
xvalues	Array of *x*-axis data values
yvalues	Array of *y*-axis data values
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Analyzes a single series of data and fills the chart environment for a single-series scatter diagram.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_analyzescatterms

short _far _pg_analyzescatterms(chartenv _far ****env*, float _far ****xvalues***,**
float _far ****yvalues*, short** ***nseries***, **short** ***n***, **short** ***rowdim***,
char _far * _far ****serieslabels* **);

Include	PGCHART.H
env	Chart environment structure
xvalues	Array of *x*-axis data values
yvalues	Array of *y*-axis data values
nseries	Number of series to chart
n	Number of data values to chart
rowdim	Row dimension of data array
serieslabels	Array of labels for series
Returns	0 if successful; a nonzero value if not

Analyzes a multiple series of data and fills the chart environment for a multiseries scatter diagram.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chart

short _far _pg_chart(chartenv _far ****env*, char _far * _far ****categories***,**
float _far ****values*, short** ***n*** **);**

Include	PGCHART.H
env	Chart environment variable
categories	Array of category variables
values	Array of data values
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Displays a single-series bar, column, or line chart, depending on the type specified in the chart environment variable.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chartms

short _far _pg_chartms(chartenv _far ****env*, char _far * _far ****categories***,**
float _far ****values*, short** ***nseries*****, short** ***n*****, short** ***arraydim*****,**
char _far * _far ****serieslabels* **);

Include	PGCHART.H
env	Chart environment variable
categories	Array of category variables
values	Array of data values
nseries	Number of series to chart
n	Number of data values to chart
arraydim	Row dimension of data array
serieslabels	Array of labels for series
Returns	0 if successful; a nonzero value if not

Displays a multiseries bar, column, or line chart, depending on the type specified in the chart environment.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chartpie

short _far _pg_chartpie(chartenv _far ****env*, char _far * _far ****categories***,**
float _far ****values*, short _far ****explode***, short** ***n*** **);**

Include	PGCHART.H
env	Chart environment structure
categories	Array of category labels
values	Array of data values
explode	Array of explode flags
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Displays a pie chart for the data contained in the array *values*. Pie charts are formed from a single series of data (there is no multiseries version of pie charts as there is for other chart types).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chartscatter

short _far _pg_chartscatter(chartenv _far ****env*, float _far ****xvalues***,**
float _far ****yvalues*, short** ***n*** **);**

Include	PGCHART.H
env	Chart environment structure
xvalues	Array of *x*-axis data values
yvalues	Array of *y*-axis data values
n	Number of data values to chart
Returns	0 if successful; a nonzero value if not

Displays a scatter diagram for a single series of data.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_chartscatterms

short _far _pg_chartscatterms(chartenv _far ****env*, float _far ****xvalues***,**
float _far ****yvalues*, short** *nseries***, short** *n***, short** *rowdim***,**
char _far * _far ****serieslabels*);**

Include	PGCHART.H
env	Chart environment structure
xvalues	Array of *x*-axis data values
yvalues	Array of *y*-axis data values
nseries	Number of series to chart
n	Number of data values to chart
rowdim	Row dimension of data array
serieslabels	Array of labels for series
Returns	0 if successful; a nonzero value if not

Displays a scatter diagram for more than one series of data.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_defaultchart

short _far _pg_defaultchart(chartenv _far ****env*, short** *charttype***, short** *chartstyle***);**

Include	PGCHART.H
env	Chart environment structure
charttype	**_PG_BARCHART, _PG_COLUMNCHART, _PG_LINECHART, _PG_SCATTERCHART, _PG_PIECHART**
chartstyle	Chart style 1 or 2
Returns	0 if successful; a nonzero value if not

Initializes all necessary variables in the chart environment for the chart type by the variable *charttype*.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_getchardef

short _far _pg_getchardef(short *charnum*, **unsigned char _far *** *chardef* **);**

Include	PGCHART.H
charnum	ASCII number of character
chardef	Pointer to 8-by-8 bit map array
Returns	0 if successful; a nonzero value if not

Retrieves the current 8-by-8 pixel bit map for the character with the ASCII number *charnum*. The bit map is stored in the *chardef* array.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_getpalette

short _far _pg_getpalette(paletteentry _far * *palette* **);**

Include	PGCHART.H
palette	Pointer to first palette structure in array
Returns	0 if successful; **_BADSCREENMODE** if current palettes have not been initialized by a previous call to **_pg_setpalette**

Retrieves palette colors, line styles, fill patterns, and plot characters for all palettes. The pointer *palette* points to an array of palette structures that will contain the desired palette values.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_getstyleset

void _far _pg_getstyleset(unsigned short _far * *styleset* **);**

Include	PGCHART.H
styleset	Pointer to current styleset
Return	No return value

Retrieves the contents of the current styleset.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_hlabelchart

short _far _pg_hlabelchart(chartenv _far ****env*, **short *x*, **short** *y*, **short** *color*,
char _far ****label* **);

Include	PGCHART.H
env	Chart environment structure
x	Pixel *x*-coordinate for text
y	Pixel *y*-coordinate for text
color	Color code for text
label	Label text
Returns	0 if successful; a nonzero value if not

Writes text horizontally on the screen. The arguments *x* and *y* are pixel coordinates for the beginning location of text relative to the upper left corner of the chart window.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_initchart

short _far _pg_initchart(void);

Include	PGCHART.H
Returns	0 if successful; a nonzero value if not

Initializes the presentation graphics package. It initializes the color and style pools, resets the chartline styleset, builds default palette modes, and reads the presentation graphics character font definition from the disk. This function is required in all programs that use presentation graphics.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_resetpalette

short _far _pg_resetpalette(void);

Include	PGCHART.H
Returns	0 if successful; **_BADSCREENMODE** if the screen mode is not valid

Sets the palette colors, line styles, fill patterns, and plot characters for the palette to the default for the current screen mode.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_resetstyleset

void _far _pg_resetstyleset(void);

Include	PGCHART.H
Returns	No return value

Reinitializes the styleset to the default values for the current screen mode.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_setchardef

short _far _pg_setchardef(short *charnum*, **unsigned char _far ***_chardef_**);**

Include	PGCHART.H
charnum	ASCII number of character
chardef	Pointer to 8-by-8 bit map array
Returns	No return value

Sets the 8-by-8 pixel bit map for the character with the ASCII number *charnum*. The bit map is stored in the *chardef* array.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_setpalette

short _far _pg_setpalette(paletteentry _far ****palette* **);

Include	PGCHART.H
palette	Pointer to first palette structure in array
Returns	0 if successful; **_BADSCREENMODE** if the new palettes are not valid

Sets palette colors, line styles, fill patterns, and plot characters for all palettes. The pointer *palette* points to an array of palette structures that contains the desired palette values.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_setstyleset

void _far _pg_setstyleset(unsigned short _far ****styleset* **);

Include	PGCHART.H
styleset	Pointer to new styleset
Returns	No return value

Sets the current styleset.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pg_vlabelchart

short _far _pg_vlabelchart(chartenv _far ****env*, **short *x*, **short** *y*, **short** *color*,
char _far ****label* **);

Include	PGCHART.H
env	Chart environment structure
x	Pixel *x*-coordinate for text
y	Pixel *y*-coordinate for text
color	Color code for text
label	Label text
Returns	0 if successful; a nonzero value if not

Writes text vertically on the screen. The arguments *x* and *y* are pixel coordinates for the beginning location of text relative to the upper left corner of the chart window.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pie

short _far _pie(short *control*, **short** *x1*, **short** *y1*, **short** *x2*, **short** *y2*, **short** *x3*, **short** *y3*,
short *x4*, **short** *y4* **);**

Include	GRAPH.H
control	**_GBORDER, _GFILLINTERIOR**
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
x3, *y3*	Start vector
x4, *y4*	End vector
Returns	A nonzero value if successful; 0 if not

Draws a pie-shaped wedge by drawing an elliptical arc whose center and two endpoints are joined by lines. Points are defined using the view coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pie_w

short _far _pie_w(short *control*, **double** *x1*, **double** *y1*, **double** *x2*, **double** *y2*, **double** *x3*, **double** *y3*, **double** *x4*, **double** *y4* **);**

Include	GRAPH.H
control	Fill-control constant
x1, *y1*	Upper left corner of bounding rectangle
x2, *y2*	Lower right corner of bounding rectangle
x3, *y3*	Start vector
x4, *y4*	End vector
Returns	A nonzero value if successful; 0 if not

Draws a pie-shaped wedge by drawing an elliptical arc whose center and two endpoints are joined by lines. Points are defined using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pie_wxy

short _far _pie_wxy(short *control*, **struct _wxycoord _far ***_pwxy1_, **struct _wxycoord _far ***_pwxy2_, **struct _wxycoord _far ***_pwxy3_, **struct _wxycoord _far ***_pwxy4_ **);**

Include	GRAPH.H
control	**_GBORDER, _GFILLINTERIOR**
pwxy1	Upper left corner of bounding rectangle
pwxy2	Lower right corner of bounding rectangle
pwxy3	Start vector
pwxy4	End vector
Returns	A nonzero value if successful; 0 if not

Draws a pie-shaped wedge by drawing an elliptical arc whose center and two endpoints are joined by lines. Points are defined using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_pipe

int _pipe(int ****phandles*, **unsigned int *psize*, **int** *textmode* **);**

Include	IO.H, FCNTL.H, ERRNO.H
phandles	Pointer to read and write handles (*phandles*[0] and *phandles*[1], respectively)
psize	Amount of memory, in bytes, reserved for the pipe
textmode	**O_BINARY, O_TEXT**
Returns	0 if successful; –1 if not
errno	**EMFILE, ENFILE**

Creates a pipe for both reading and writing, generally in preparation for linking it to a child process.

☐ ANSI ☐ DOS ■ OS/2 ■ UNIX ■ XENIX

_polygon

short _far _polygon(short *control*, **struct xycoord _far ****points*, **short** *numpoints* **);**

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
points	Array of **xycoord** structures specifying the polygon's vertices
numpoints	Number of vertices
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws or scan-fills a polygon, using the view coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_polygon_w

short _far _polygon_w(short *control*, **double _far *** *points*, **short** *numpoints* **);**

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
points	Values specifying the polygon's vertices
numpoints	Number of vertices
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws or scan-fills a polygon, using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_polygon_wxy

short _far _polygon_wxy(short *control*, **struct _wxycoord _far *** *points*,
short *numpoints* **);**

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
points	Array of **_wxycoord** structures specifying the polygon's vertices
numpoints	Number of vertices
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws or scan-fills a polygon, using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_popen

FILE * _popen(char *_command_, char *_mode_);

Include	STDIO.H
command	Command string
mode	Access permissions: **"r", "rt", "rb", "w", "wt", "wb"**
Returns	A stream associated with one end of the pipe if successful; **NULL** if not

Creates a pipe and asynchronously executes a child copy of the command processor.

□ ANSI □ DOS ■ OS/2 ■ UNIX ■ XENIX

pow

double pow(double _x_, double _y_);

Include	MATH.H, ERRNO.H
x	Number to be raised
y	Power of *x*
Returns	The value of x^y; 1 if *x* is not 0 and *y* is 0; ± **HUGE_VAL** if *x* is 0 and *y* is negative; 0 if both *x* and *y* are 0 or if *x* is negative and *y* is not an integer
errno	**EDOM, ERANGE**

Computes the value of x^y.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

powl

long double powl(long double *x*, **long double** *y* **);**

Include	MATH.H, ERRNO.H
x	Number to be raised
y	Power of *x*
Returns	The value of x^y; 1 if *x* is not 0 and *y* is 0; ± **_LHUGE_VAL** if *x* is 0 and *y* is negative; 0 if both *x* and *y* are 0 or if *x* is negative and *y* is not an integer
errno	**EDOM, ERANGE**

Computes the value of x^y. Long double-precision floating-point version of **pow**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

printf

int printf(const char ***_format_ [[, _argument_]]... **);

Include	STDIO.H
format	Format-control string
argument	Optional arguments, variables of the type specified in the format-control string
Returns	The number of characters printed if successful; a negative value if not

Formats and prints a series of characters and values to the standard output stream, **stdout**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

putc

int putc(int *c*, **FILE ***_stream_ **);**

Include	STDIO.H
c	Character to be written
stream	Pointer to target stream
Returns	The character written if successful; **EOF** may indicate failure

Writes the single character *c* to the output *stream* at the current position.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

putch

int putch(int *c* **);**

Include	CONIO.H
c	Character to be output
Returns	*c* if successful; **EOF** if not

Writes the character *c* directly to the console.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

putchar

int putchar(int *c* **);**

Include	STDIO.H
c	Character to be written
Returns	The character written if successful; **EOF** may indicate failure

Writes the single character *c* to the standard output stream, **stdout**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

putenv

int putenv(char ****envstring* **);

Include	STDLIB.H
envstring	Environment-string definition
Returns	0 if successful; –1 if not

Adds or removes an environment variable or modifies the value of an existing one.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_putimage

void _far _putimage(short *x*, **short** *y*, **char _huge ****image*, **short** *action* **);**

Include	GRAPH.H
x, *y*	Position of upper left corner of image
image	Stored image buffer
action	**_GAND, _GOR, _GPRESET, _GPSET, _GXOR**
Returns	No return value; check **_grstatus**

Transfers to the screen the image stored in the buffer *image*, placing the upper left corner of the image at the view coordinate (*x*, *y*).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_putimage_w

void _far _putimage_w(double *wx*, **double** *wy*, **char _huge ***_image_, **short** *action* **);**

Include	GRAPH.H
wx, *wy*	Position of upper left corner of image
image	Stored image buffer
action	**_GAND, _GOR, _GPRESET, _GPSET, _GXOR**
Returns	No return value; check **_grstatus**

Transfers to the screen the image stored in the buffer *image*, placing the upper left corner of the image at the window coordinate (*wx*, *wy*).

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

puts

int puts(const char ***_string_ **);

Include	STDIO.H
string	String to be output
Returns	A nonnegative value if successful; **EOF** if not

Writes *string* to the standard output stream, **stdout**, replacing the string's terminating null character ('\0') with a newline character (\n) in the output stream.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

putw

int putw(int *binint*, **FILE** ***stream*);**

Include	STDIO.H
binint	Binary integer to be output
stream	Pointer to target stream
Returns	The value written if successful; **EOF** may indicate an error

Writes a binary value of type **int** to the current position of the *stream*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

qsort

void qsort(void ***base*, size_t** *num*, **size_t** *width*,
int(***compare*)(const void** ***elem1*, const void** ***elem2*));**

Include	STDLIB.H, SEARCH.H
base	Start of array
num	Number of elements in array
width	Element size in bytes
compare	Comparison function
elem1	Pointer to key for the search
elem2	Pointer to the array element to be compared with the key
Returns	No return value

Implements a quick-sort algorithm to sort an array of *num* elements, each of *width* bytes.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

raise

int raise(int *sig* **);**

Include	SIGNAL.H
sig	**SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGUSR3, SIGBREAK**
Returns	0 if successful; a nonzero value if not

Raises the *sig* signal condition in the executing program.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

rand

int rand(void);

Include	STDLIB.H
Returns	A pseudorandom integer in the range 0 to **RAND_MAX**

Generates a pseudorandom number.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

read

int read(int *handle*, **void *** *buffer*, **unsigned int** *count* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
buffer	Storage location for data
count	Maximum number of bytes
Returns	The number of bytes read if successful; –1 if not
errno	**EBADF**

Attempts to read *count* bytes into *buffer* from the file associated with *handle*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

realloc

void *realloc(void *_memblock_, size_t _size_);

Include	STDLIB.H or MALLOC.H
memblock	Pointer to previously allocated memory block
size	New size in bytes
Returns	A **void** pointer to the reallocated memory block if successful; **NULL** if not

Changes the size and, possibly, the location of a previously allocated memory block.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_rectangle

short _far _rectangle(short _control_, short _x1_, short _y1_, short _x2_, short _y2_);

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
x1, *y1*	Upper left corner
x2, *y2*	Lower right corner
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws a rectangle with the current color, line style, and write mode, using the view coordinate system.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_rectangle_w

short _far _rectangle_w(short *control*, **double** *wx1*, **double** *wy1*,
double *wx2*, **double** *wy2* **);**

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
wx1, *wy1*	Upper left corner
wx2, *wy2*	Lower right corner
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws a rectangle with the current color, line style, and write mode, using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_rectangle_wxy

short _far _rectangle_wxy(short *control*, **struct _wxycoord _far *** *pwxy1*,
struct _wxycoord _far * *pwxy2* **);**

Include	GRAPH.H
control	**_GFILLINTERIOR, _GBORDER**
pwxy1	Upper left corner
pwxy2	Lower right corner
Returns	A nonzero value if anything is drawn; 0 if nothing is drawn

Draws a rectangle with the current color, line style, and write mode, using the window coordinate system.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_registerfonts

short _far _registerfonts(unsigned char _far ****pathname* **);

Include	GRAPH.H
pathname	Path name specifying .FON files to be registered
Returns	The number of fonts registered if successful; a negative value if not

Initializes the font graphics system. Font files must be registered with the **_registerfonts** function before any other font-related library function can be used.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_remapallpalette

short _far _remapallpalette(long _far ****colors* **):

Include	GRAPH.H
colors	Array of color values
Returns	A nonzero value if successful; 0 if not

Remaps all of the available colors simultaneously.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_remappalette

long _far _remappalette(short *index*, **long** *color* **);**

Include	GRAPH.H
index	Color index to redefine
color	Color value to assign to *index*
Returns	The previous color value of *index* if successful; –1 if not

Remaps the specified color index to *color*, which must be a color value supported by the current video configuration.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

remove

int remove(const char ***_filename_);**

Include	STDIO.H or IO.H, ERRNO.H
filename	Path name of file to be removed
Returns	0 if successful; a nonzero value if not
errno	**EACCESS, ENOENT**

Deletes the file specified by *filename.*

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

rename

int rename(const char ***_oldname_, const char ***_newname_**);**

Include	STDIO.H or IO.H, ERRNO.H
oldname	Pointer to old name
newname	Pointer to new name
Returns	0 if successful; a nonzero value if not
errno	**EACCESS, ENOENT, EXDEV**

Renames the file or directory specified by *oldname* to the name given by *newname.*

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

rewind

void rewind(FILE ***_stream_);**

Include	STDIO.H
stream	Pointer to target stream
Returns	No return value

Repositions the file pointer associated with *stream* to the beginning of the file.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

rmdir

int rmdir(char *_dirname_);

Include	DIRECT.H, ERRNO.H
dirname	Path name of directory to be removed
Returns	0 if successful; a nonzero value if not
errno	**EACCESS, ENOENT**

Deletes the directory specified by *dirname*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

rmtmp

int rmtmp(void);

Include	STDIO.H
Returns	The number of temporary files closed and deleted

Cleans up all the temporary files in the current directory. The function removes only those files created by **tmpfile** and should be used only in the same directory in which the temporary files were created.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_rotl

unsigned int _rotl(unsigned int *value*, **int** *shift* **);**

Include	STDLIB.H
value	Value to be rotated
shift	Number of bits to shift
Returns	*value* rotated to the left by *shift* bits

Rotates bits to the left.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_rotr

unsigned int _rotr(unsigned int *value*, **int** *shift* **);**

Include	STDLIB.H
value	Value to be rotated
shift	Number of bits to shift
Returns	*value* rotated to the right by *shift* bits

Rotates bits to the right.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

scanf

int scanf(const char ***_format_ [[, _argument_]]... **);

Include	STDIO.H
format	Format-control string
argument	Optional arguments; pointers to variables of the types specified in the format-control string
Returns	The number of fields successfully converted and assigned; **EOF** if input failure occurs before any conversion

Reads data from the standard input stream, **stdin**, into the locations given by an argument.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_scrolltextwindow

void _far _scrolltextwindow(short *lines* **);**

Include	GRAPH.H
lines	Number of lines to scroll
Returns	No return value

Scrolls the current text window. A positive value for *lines* scrolls the window up, and a negative value scrolls down.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_searchenv

void _searchenv(char ***_filename_, char ***_varname_**, char ***_pathname_ **);**

Include	STDLIB.H
filename	Name of file to search for
varname	Environment to search
pathname	Buffer to store complete path
Returns	No return value

Searches for the target file in the specified *varname* domain.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

segread

void segread(struct SREGS ***_segregs_ **);

Include	DOS.H
segregs	Segment-register values
Returns	No return value

Fills the structure pointed to by *segregs* with the current contents of the segment registers.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_selectpalette

short _far _selectpalette(short *number* **);**

Include	GRAPH.H
number	Palette number
Returns	The previous palette number; –1 if an error occurs

Selects one of several predefined palettes, depending on video mode and hardware (**_MRES4COLOR**, **_MRESNOCOLOR**, and **_ORESCOLOR** modes only).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setactivepage

short _far _setactivepage(short *page* **);**

Include	GRAPH.H
page	Memory page number
Returns	The number of the previous active page if successful; –1 if an error occurs

Specifies the area in memory where output is written for hardware and mode configurations with enough memory to support multiple-screen pages. Note that OS/2 supports only one page. Page 0 is always supported.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_setbkcolor

long _far _setbkcolor(long *color* **);**

Include	GRAPH.H
color	Desired background color
Returns	The previous background color

Sets the current background color to *color*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

setbuf

void setbuf(FILE ***_stream_, char ***_buffer_**);**

Include	STDIO.H
stream	Pointer to target stream
buffer	User-allocated buffer of size **BUFSIZ** or **NULL** for no buffering
Returns	No return value

Allows the user to assign a buffer or disable buffering for *stream*. The argument *stream* must refer to an open file before it has been read or written.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_setcliprgn

void _far _setcliprgn(short *x1*, **short** *y1*, **short** *x2*, **short** *y2* **);**

Include	GRAPH.H
x1, *y1*	Upper left corner of clip region
x2, *y2*	Lower right corner of clip region
Returns	No return value; check **_grstatus**

Limits the display of subsequent graphics output and font text output to the part that fits within a particular area of the screen, known as the clipping region.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_setcolor

short _far _setcolor(short *color* **);**

Include	GRAPH.H
color	Desired color index
Returns	The previous color index if successful; –1 if not

Sets the current color index (used by graphics and font text output) to *color*.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

_setfillmask

void _far _setfillmask(unsigned char _far *_mask_);

Include	GRAPH.H
mask	Mask array
Returns	No return value

Sets the current fill mask. The mask is an 8-by-8 array of bits, where each bit represents a pixel. A 1 bit sets the corresponding pixel to the current color, whereas a 0 bit leaves the pixel unchanged.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setfont

short _far _setfont(unsigned char _far *_options_);

Include	GRAPH.H
options	String describing font characteristics
Returns	The font index number if successful; a negative value if not

Finds a single font from the set of registered fonts that has the characteristics specified by the *options* string and returns the index of that font.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setgtextvector

struct xycoord _far _setgtextvector(short _x_, short _y_);

Include	GRAPH.H
x, y	Values defining a vector that determines the direction of font text rotation
Returns	The previous vector in an **xycoord** structure defined in GRAPH.H

Sets the current orientation for font text output. (0, 0) is ignored.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

setjmp

int setjmp(jmp_buf *env* **);**

Include	SETJMP.H
env	Variable in which environment is stored
Returns	0 after saving the stack environment; the value argument of **longjmp** if **setjmp** returns as a result of a **longjmp** call; 1 if the **longjmp** argument is 0

Saves a stack environment that can subsequently be restored using **longjmp**. Used together this way, **setjmp** and **longjmp** provide a way to execute a nonlocal goto.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_setlinestyle

void _far _setlinestyle(unsigned short *mask* **);**

Include	GRAPH.H
mask	Desired line-style mask
Returns	No return value

Selects the mask used for line drawing. The mask is a 16-bit number, where each bit represents a pixel in the line being drawn. If a bit is 1, the corresponding pixel is set to the current color. If a bit is 0, the corresponding pixel is left unchanged.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

setlocale

char *setlocale(int *category*, **const char ***locale* **);**

Include	LOCALE.H
category	**LC_ALL, LC_COLLATE, LC_TYPE, LC_MONETARY, LC_NUMERIC, LC_TIME, NULL**
locale	"C"
Returns	A pointer to the string associated with *category* for the new locale, if valid; a null pointer if invalid

Sets categories specified for a certain location.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

setmode

int setmode(int *handle*, **int** *mode* **);**

Include	FCNTL.H, IO.H, ERRNO.H
handle	File handle
mode	**O_TEXT, O_BINARY**
Returns	The previous translation mode if successful; –1 if not
errno	**EBADF, EINVAL**

Sets the translation mode of the file given by *handle* to *mode*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_setpixel

short _far _setpixel(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	Target pixel
Returns	The previous value of the target pixel if successful; –1 if not

Sets a pixel at the specified view coordinate location to the current color.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setpixel_w

short _far _setpixel_w(double *wx*, **double** *wy* **);**

Include	GRAPH.H
wx, *wy*	Target pixel
Returns	The previous value of the target pixel if successful; –1 if not

Sets a pixel at the specified window coordinate location to the current color.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_settextcolor

short _far _settextcolor(short *index* **);**

Include	GRAPH.H
index	Desired color index
Returns	The color index of the previous text attribute

Sets the current text attribute to the color index specified by *index*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_settextcursor

short _far _settextcursor(short *attr* **);**

Include	GRAPH.H
attr	0x0707, 0x0007, 0x0607, 0x2000, etc.
Returns	The previous cursor attribute if successful; –1 if not

In text modes, sets the BIOS cursor attribute (the shape) to the value specified by *attr*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_settextposition

struct rccoord _far _settextposition(short *row*, **short** *column* **);**

Include	GRAPH.H
row, *column*	New text output start position
Returns	The previous text position

Sets the current text position to the location (*row*, *column*) within the current text window.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_settextrows

short _far _settextrows(short *rows* **);**

Include	GRAPH.H
rows	Number of text rows requested
Returns	The number of rows actually set if successful; 0 if an unexpected error occurs

Requests the number of text rows to be used in the current video mode.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_settextwindow

void _far _settextwindow(short *r1*, **short** *c1*, **short** *r2*, **short** *c2* **);**

Include	GRAPH.H
r1, *c1*	Upper left corner of window
r2, *c2*	Lower right corner of window
Returns	No return value; check **_grstatus**

Specifies a window in row and column coordinates where all the text output to the screen is confined.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

setvbuf

int setvbuf(FILE ***_stream_, **char ***_buffer_, **int *mode*, **size_t** *size* **);**

Include	STDIO.H
stream	Pointer to target stream
buffer	User-supplied buffer or **NULL**
mode	**_IOFBF**, **_IOLBF**, **_IONBF**
size	Size of buffer
Returns	0 if successful; a nonzero value if not

Allows the user to control both buffering and buffer size for *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_setvideomode

short _far _setvideomode(short *mode* **);**

Include	GRAPH.H
mode	**_MAXRESMODE, _MAXCOLORMODE, _DEFAULTMODE, _TEXTBW40, _TEXTC40, _TEXTBW80, _TEXTC80, _MRES4COLOR, _MRESNOCOLOR, _HRESBW, _TEXTMONO, _HERCMONO, _MRES16COLOR, _HRES16COLOR, _ERESNOCOLOR, _ERESCOLOR, _VRES2COLOR, _VRES16COLOR, _MRES256COLOR, _ORESCOLOR**
Returns	The number of text rows if successful; 0 if not

Selects a screen mode appropriate for a particular hardware/display configuration. Only text modes are available in OS/2.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_setvideomoderows

short _far _setvideomoderows(short *mode*, **short** *rows* **);**

Include	GRAPH.H
mode	**_MAXRESMODE, _MAXCOLORMODE, _DEFAULTMODE, _TEXTBW40, _TEXTC40, _TEXTBW80, _TEXTC80, _MRES4COLOR, _MRESNOCOLOR, _HRESBW, _TEXTMONO, _HERCMONO, _MRES16COLOR, _HRES16COLOR, _ERESNOCOLOR, _ERESCOLOR, _VRES2COLOR, _VRES16COLOR, _MRES256COLOR, _ORESCOLOR**
rows	Number of text rows requested
Returns	The number of rows actually set if successful; 0 if not

Selects a screen mode for a particular hardware/display combination, and requests the number of text rows to be used. Only text modes are available in OS/2.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_setvieworg

struct xycoord _far _setvieworg(short *x*, **short** *y* **);**

Include	GRAPH.H
x, *y*	New origin point
Returns	The physical coordinates of the previous view origin

Moves the viewport origin (0, 0) to the physical point (*x*, *y*). (All other view coordinate points move the same direction and distance.)

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setviewport

void _far _setviewport(short *x1*, **short** *y1*, **short** *x2*, **short** *y2* **);**

Include	GRAPH.H
x1, *y1*	Upper left corner of viewport
x2, *y2*	Lower right corner of viewport
Returns	No return value; check **_grstatus**

Redefines the graphics viewport (the active drawing area within the boundaries of the physical screen).

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setvisualpage

short _far _setvisualpage(short *page* **);**

Include	GRAPH.H
page	Visual page number
Returns	The number of the previous visual page if successful; a negative value if not

Specifies the area in memory where output is displayed for hardware and mode configurations with enough memory to support multiple-screen pages. Note that OS/2 supports only one page. Page 0 is always supported.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_setwindow

short _far _setwindow(short *finvert*, **double** *wx1*, **double** *wy1*,
double *wx2*, **double** *wy2* **);**

Include	GRAPH.H
finvert	Invert flag
wx1, *wy1*	Upper left corner of window
wx2, *wy2*	Lower right corner of window
Returns	A nonzero value if successful; 0 if not

Defines a virtual window bounded by the specified coordinates within the current viewport. All *w* and *wxy* drawing functions produce output relative to this window.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

_setwritemode

short _far _setwritemode(short *action* **);**

Include	GRAPH.H
action	**_GPSET, _GPRESET, _GAND, _GOR, _GXOR**
Returns	The previous write mode; –1 if an error occurs

Sets the current logical write mode used when drawing lines with the **_lineto**, **_rectangle**, and **_polygon** functions.

□ ANSI ■ DOS □ OS/2 □ UNIX □ XENIX

signal

void(*signal(int *sig*, **void(***func*)**(int** *sig* [[, **int** *subcode*]])))(**int** *sig* **);**

Include	SIGNAL.H, ERRNO.H
sig	**SIGABRT, SIGBREAK, SIGFPE, SIGILL, SIGINT, SIGSEGV, SIGTERM, SIGUSR1, SIGUSR2, SIGUSR3**
func	Function address or one of **SIG_IGN, SIG_DFL, SIG_SGE, SIG_ACK**
subcode	Optional subcode to the signal number
Returns	The previous value of *func* associated with the given signal if successful; –1 if not
errno	**EINVAL**

Allows a process to choose one of several ways to handle an interrupt signal from the operating system.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sin

double sin(double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The sine of *x*
errno	**ERANGE**

Calculates the sine of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sinh

double sinh(double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The hyperbolic sine of *x*; ± **HUGE_VAL** if not
errno	**ERANGE**

Calculates the hyperbolic sine of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sinhl

long double sinhl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The hyperbolic sine of *x* if successful; ± **_LHUGE_VAL** if not
errno	**ERANGE**

Calculates the hyperbolic sine of *x*. Long double-precision floating-point version of **sinh**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

sinl

long double sinl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The sine of *x*
errno	**ERANGE**

Calculates the sine of *x*. Long double-precision floating-point version of **sin**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

sopen

int sopen(char ***_filename_, int** *oflag*, **int** *shflag* [[, **int** *pmode*]] **);**

Include	SYS\TYPES.H, SYS\STAT.H, FCNTL.H, SHARE.H, IO.H
filename	File path name
oflag	**O_APPEND, O_BINARY, O_CREAT, O_EXCL, O_RDONLY, O_RDWR, O_TEXT, O_TRUNC, O_WRONLY**
shflag	**SH_COMPAT, SH_DENYRW, SH_DENYWR, SH_DENYRD, SH_DENYNO**
pmode	**S_IWRITE, S_IREAD, S_IREAD | S_IWRITE**
Returns	A file handle for the opened file if successful; –1 if not
errno	**EACCES, EEXIST, EINVAL, EMFILE, ENOENT**

Opens the file specified by *filename* and prepares the file for subsequent shared reading or writing, as defined by *oflag*, *shflag*, and, optionally, *pmode*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnl

int spawnl(int *mode*, **char *** *cmdname*, **char *** *arg0*, **char *** *arg1*, **... char *** *argn*, **NULL);**

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process, and passes a copy of the current process's environment to the child. Command-line arguments are passed individually to the function.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnle

int spawnle(int *mode*, **char *** *cmdname*, **char *** *arg0*, **char *** *arg1*, **... char *** *argn*, **NULL, char **** *envp* **);**

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
envp	Array of pointers to environment settings
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process. Command-line arguments are passed individually to the function. The child process is passed an array of pointers to environment strings instead of a copy of the current process's environment strings.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnlp

int spawnlp(int *mode*, **char *** *cmdname*, **char *** *arg0*, **char *** *arg1*, **... char *** *argn*, **NULL);**

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
arg0, ... *argn*	List of pointers to arguments, terminated by **NULL**
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process, and passes to the child a copy of the environment of the current process. Command-line arguments are passed individually to the function. The PATH environment variable is used to find the file to be executed if a relative path name is given.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

spawnlpe

int spawnlpe(int *mode*, **char *** *cmdname*, **char *** *arg0*, **char *** *arg1*, **... char *** *argn*, **NULL, char **** *envp* **);**

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
arg0, ... argn	List of pointers to arguments, terminated by **NULL**
envp	Array of pointers to environment settings
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process. Command-line arguments are passed individually to the function. The PATH environment variable is used to find the file to be executed if it is specified by a relative path name. The child process is passed an array of pointers to environment strings, instead of a copy of the environment strings of the current process.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

spawnv

int spawnv(int *mode*, **char *** *cmdname*, **char **** *argv* **);**

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process, and passes a copy of the current process's environment to the child. Command-line arguments are passed to the function as an array of pointers.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

spawnve

int spawnve(int *mode*, **char *** *cmdname*, **char **** *argv* , **char **** *envp* **);**

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
envp	Array of pointers to environment settings
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process. Command-line arguments are passed to the function as an array of pointers. The child process is passed an array of pointers to environment strings, instead of a copy of the environment strings of the current process.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

spawnvp

int spawnvp(int *mode*, **char *** *cmdname*, **char **** *argv* **);**

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process, and passes a copy of the current process's environment to the child. Command-line arguments are passed to the function as an array of pointers. The PATH environment variable is used to find the file to be executed if necessary and if the file is specified by a relative path name.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

spawnvpe

int spawnvpe(int *mode*, **char *** *cmdname*, **char **** *argv*, **char **** *envp* **);**

Include	STDIO.H, PROCESS.H, ERRNO.H
mode	**P_WAIT, P_NOWAIT, P_NOWAITO, P_OVERLAY, P_DETACH**
cmdname	Path name of file to be executed
argv	Array of pointers to arguments
envp	Array of pointers to environment settings
Returns	The exit status of the child process (synchronous spawn) or the process ID of the child process (asynchronous spawn)
errno	**E2BIG, EINVAL, ENOENT, ENOEXEC, ENOMEM**

Creates and executes a new child process. Command-line arguments are passed to the function as an array of pointers. The PATH environment variable is used to find the file to be executed. It explicitly passes to the child process the strings, instead of a copy of the current process's environment strings array of pointers to environment.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_splitpath

void _splitpath(char * *path*, **char *** *drive*, **char *** *dir*, **char *** *fname*, **char *** *ext* **);**

Include	STDLIB.H
path	Full path-name buffer
drive	Drive letter
dir	Directory path
fname	File name
ext	File extension
Returns	No return value

Breaks the full path-name buffer *path* into its four components.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

sprintf

int sprintf(char ***_buffer_, const char ***_format_ [[, _argument_]]...);

Include	STDIO.H
buffer	Storage location for output
format	Format-control string
argument	Optional arguments; pointers to variables of the types specified in the format-control string
Returns	The number of characters stored in *buffer*, not counting the terminating null character (**'\0'**)

Formats and stores a series of characters and values in *buffer*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sqrt

double sqrt(double *x* **);**

Include	MATH.H, ERRNO.H
x	Nonnegative floating-point value
Returns	The square root of *x* if successful; 0 if not
errno	**EDOM**

Calculates the square root.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sqrtl

long double sqrtl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Nonnegative long double-precision floating-point value
Returns	The square root of *x* if successful; 0 if not
errno	**EDOM**

Calculates the square root. Long double-precision floating-point version of **sqrt**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

srand

void srand(unsigned int *seed* **);**

Include	STDLIB.H
seed	Seed for random-number generation
Returns	No return value

Sets the starting point for generating a series of pseudorandom integers.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

sscanf

int sscanf(const char ***_buffer_, const char ***_format_ [[, _argument_]]...);

Include	STDIO.H
buffer	Stored data
format	Format-control string
argument	Optional arguments; pointers to variables of types specified in the format-control string
Returns	The number of fields that were successfully converted and assigned

Reads data from *buffer* into the locations given by each argument. Every argument must be a pointer to a variable with a type that corresponds to a type specifier in *format*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

stackavail

size_t stackavail(void);

Include	MALLOC.H
Returns	The size in bytes as an unsigned integer

Gets the approximate size in bytes of the stack space available for dynamic memory allocation with **alloca**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

stat

int stat(char ****pathname*, **struct stat ****buffer* **);

Include	SYS\STAT.H, SYS\TYPES.H, ERRNO.H
pathname	Path name of existing file
buffer	Pointer to structure to receive results
Returns	0 if successful; a nonzero value if not
errno	**ENOENT**

Gets information about the file or directory specified by *pathname* and stores it in the structure that *buffer* points to.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_status87

unsigned int _status87(void);

Include	FLOAT.H
Returns	A floating-point status word whose bits are defined in FLOAT.H

Gets the current 8087 floating-point status word.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strcat

char *strcat(char ****string1*, **const char ****string2* **);

Include	STRING.H
string1	Destination string
string2	Source string
Returns	A pointer to the beginning of the concatenated string

Appends *string2* to *string1*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strchr

char *strchr(const char *_string_, int _c_);

Include	STRING.H
string	Source string
c	Character to be located
Returns	A pointer to the first occurrence of *c* in *string*

Searches for the first occurrence of *c* in *string*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strcmp

int strcmp(const char *_string1_, const char *_string2_);

Include	STRING.H
string1, *string2*	Strings to compare
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares two strings.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strcoll

int strcoll(const char *_string1_, const char *_string2_);

Include	STRING.H
string1, *string2*	Strings to compare
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares two strings for locale-specific collating sequences.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strcpy

char *strcpy(char *_string1_, const char *_string2_);

Include	STRING.H
string1	Destination string
string2	Source string
Returns	*string1*

Copies *string2* to *string1*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strcspn

size_t strcspn(const char *_string1_, const char *_string2_);

Include	STRING.H
string1	Source string
string2	Character set
Returns	The length of the maximum initial segment of *string1* that consists of characters not in *string2*

Finds first substring in *string1* of characters not in *string2*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_strdate

char *_strdate(char *_datestr_);

Include	TIME.H
datestr	Current date
Returns	A pointer to the date

Copies the current date to the buffer that *datestr* points to, formatted as mm/dd/yy.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strdup

char *strdup(const char *_string_);

Include	STRING.H
string	Source string
Returns	A pointer to the storage space containing the duplicate string

Duplicates *string*, copying it to memory allocated by **malloc**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

strerror

char *strerror(int *errnum* **);**

Include	STRING.H
errnum	Error number
Returns	A pointer to the error-message string

Maps *errnum* to an error-message string, returning a pointer to the string.

■ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

_strerror

char *_strerror(char *_string_);

Include	STRING.H
string	User-supplied message
Returns	No return value

Returns a pointer to a string containing, in order, the string message, a colon, a space, the system error message for the last library call producing an error, and a newline character. The supplied string message can be a maximum of 94 bytes long.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

strftime

size_t strftime(char ***_string_, size_t** _maxsize_**, const char ***_format_**,**
const struct tm ***_timeptr_ **);

Include	TIME.H
string	Output string
maxsize	Maximum length of string
format	Format-control string
timeptr	**tm** data structure
Returns	The number of characters placed in *string* if the total number, including the terminating null character (**'\0'**), is less than *maxsize*; 0 otherwise

Copies text into a string related to date and time values from a **tm** structure and specified by *format*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

stricmp

int stricmp(const char ***_string1_, const char ***_string2_ **);**

Include	STRING.H
string1, *string2*	Strings to compare
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares two strings without regard to case.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strlen

size_t strlen(const char *_string_);

Include	STRING.H
string	Null-terminated string
Returns	The length in bytes of *string*, not including the terminating null character ('\0')

Gets the length of a string.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strlwr

char *strlwr(char *_string_);

Include	STRING.H
string	String to be converted
Returns	A pointer to the converted string

Converts any uppercase letters in the given null-terminated *string* to lowercase. Other characters are not affected.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strncat

char *strncat(char *_string1_, const char *_string2_, size_t _count_);

Include	STRING.H
string1	Destination string
string2	Source string
count	Maximum number of characters to append
Returns	A pointer to the concatenated string

Appends, at most, the first *count* characters of *string2* to *string1* and terminates the resulting string with a null character ('\0').

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strncmp

int strncmp(const char *_string1_, const char *_string2_, size_t _count_);

Include	STRING.H
string1, *string2*	Strings to compare
count	Maximum number of characters to compare
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares, at most, the first *count* characters of *string1* and *string2*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strncpy

char *strncpy(char *_string1_, const char *_string2_, size_t _count_);

Include	STRING.H
string1	Destination string
string2	Source string
count	Number of characters copied
Returns	*string1*

Copies, at most, *count* characters of *string2* to *string1*. If *string2* is shorter than *count*, null ('\0') characters are appended to *string1*. Warning: If the sum of the lengths of *string1* and *string2* is longer than *count*, no null character will be appended.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strnicmp

int strnicmp(const char *_string1_, const char *_string2_, size_t _count_);

Include	STRING.H
string1, *string2*	Strings to compare
count	Maximum number of characters compared
Returns	A value less than, equal to, or greater than 0, depending on whether the string pointed to by *string1* is less than, equal to, or greater than the string pointed to by *string2*

Compares, at most, the first *count* characters of *string1* and *string2*, without regard to letter case.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strnset

char *strnset(char ***_string_, int** _c_**, size_t** _count_ **);**

Include	STRING.H
string	String to be initialized
c	Character setting
count	Maximum number of characters set
Returns	A pointer to the altered string

Sets, at most, the first *count* characters of *string* to the character *c*. If *count* is longer than *string*, the length of *string* replaces *count*.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

strpbrk

char *strpbrk(const char ***_string1_, const char ***_string2_ **);**

Include	STRING.H
string1	Source string
string2	Character set
Returns	A pointer to the first occurrence of any character from *string2* in *string1*

Finds the first occurrence in *string1* of any character from *string2*, not including the terminating null character (**'\0'**).

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strrchr

char *strrchr(const char *_string_, int _c_);

Include	STRING.H
string	String to be searched
c	Character to be located
Returns	A pointer to the last occurrence of *c* in *string*; a null pointer if *c* is not found

Finds the last occurrence of the character *c* in *string*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strrev

char *strrev(char *_string_);

Include	STRING.H
string	String to be reversed
Returns	A pointer to the altered string

Reverses the order of the characters in *string*. The terminating null character (**'\0'**) remains in place.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strset

char *strset(char *_string_, int _c_);

Include	STRING.H
string	String to be set
c	Character setting
Returns	A pointer to the altered string

Sets all of the characters of *string* to *c*, except the terminating null character (**'\0'**).

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strspn

size_t strspn(const char ****string1*, const char ****string2***);**

Include	STRING.H
string1	Searched string
string2	Character set
Returns	The length of the initial substring of *string1* that consists entirely of characters from *string2*

Finds the first substring and returns the index of the first character in *string1* that does not belong to the set of characters specified by *string2*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strstr

char *strstr(const char ****string1*, const char ****string2***);**

Include	STRING.H
string1	Searched string
string2	String to search for
Returns	A pointer to the first occurrence of *string2* in *string1*; **NULL** if *string2* is not found

Finds a substring.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

_strtime

char *_strtime(char ****timestr*);**

Include	TIME.H
timestr	Time string
Returns	A formatted string

Copies the current time into the buffer that *timestr* points to, formatted as hh:mm:ss.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strtod

double strtod(const char *nptr, char **endptr);

Include	STDLIB.H, ERRNO.H
nptr	String to convert
endptr	Pointer to the character causing the end of scan
Returns	The value of the floating-point number if successful; ± **HUGE_VAL** on overflow; 0 otherwise
errno	**ERANGE**

Converts a character string to a double-precision value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strtok

char *strtok(char *string1, const char *string2);

Include	STRING.H
string1	String containing token(s)
string2	Set of delimiter characters
Returns	A pointer to the first token in *string1*; successive calls against the same string return the subsequent tokens in the string

Reads *string1* as a series of zero or more tokens and *string2* as the set of characters serving as delimiters of the tokens in *string1*. Inserts a null character each time it is called, modifying *string1*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strtol

long strtol(const char ****nptr*, **char *****endptr*, **int *base*);**

Include	STDLIB.H, ERRNO.H
nptr	String to convert
endptr	Pointer to the character causing the end of scan
base	Number base to use
Returns	The value represented in the string if successful; **LONG_MAX** or **LONG_MIN** on overflow; 0 otherwise
errno	**ERANGE**

Converts a character string to a long-integer value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_strtold

long double _strtold(const char ****nptr*, **char *****endptr*);

Include	STDLIB.H, ERRNO.H
nptr	String to convert
endptr	Pointer to the character causing the end of scan
Returns	The long double-precision value of the floating-point number if successful; ± **_LHUGE_VAL** on overflow; 0 otherwise
errno	**ERANGE**

Converts a character string to a long double-precision floating-point value.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

strtoul

unsigned long strtoul(const char ***_nptr_, char ****_endptr_**, int** _base_ **);**

Include	STDLIB.H, ERRNO.H
nptr	String to convert
endptr	Pointer to the character causing the end of scan
base	Number base to use
Returns	The converted value if successful; **ULONG_MAX** on overflow; 0 otherwise
errno	**ERANGE**

Converts a character string to an unsigned-long-integer value.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strupr

char *strupr(char ***_string_ **);

Include	STRING.H
string	String to be capitalized
Returns	A pointer to the converted string

Converts any lowercase letters in *string* to uppercase. Other characters are not affected.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

strxfrm

size_t strxfrm(char ****string1*, const char ****string2***, size_t** *count* **);**

Include	STRING.H
string1	Result string
string2	Source string
count	Number of characters transformed
Returns	The length of the transformed string, excluding the terminating null character (**'\0'**).

Transforms the string pointed to by *string2* into a new form, based on locale-specific information and stores the result in *string1*.

■ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

swab

void swab(char ****source*, char ****dest***, int** *n* **);**

Include	STDLIB.H
source	Data to be copied and swapped
dest	Storage location for swapped data
n	Number of bytes copied
Returns	No return value

Copies *n* bytes from *source*, swaps each pair of adjacent bytes, and stores the result at *dest*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

system

int system(const char ****command* **);

Include	PROCESS.H, STDLIB.H, ERRNO.H
command	Command to be executed
Returns	A nonzero value if *command* is **NULL** and the command interpreter is found; otherwise, the return code from spawning the command interpreter
errno	**E2BIG, ENOENT, ENOEXEC, ENOMEM**

Passes *command* to the command interpreter and executes the string as an operating system command.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tan

double tan(double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The tangent of *x* if successful; 0 if not
errno	**ERANGE**

Calculates the tangent of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tanh

double tanh(double *x* **);**

Include	MATH.H
x	Angle in radians
Returns	The hyperbolic tangent of *x*

Calculates the hyperbolic tangent of *x*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tanhl

long double tanhl(long double *x* **);**

Include	MATH.H
x	Angle in radians
Returns	The hyperbolic tangent of *x*

Calculates the hyperbolic tangent of *x*. Long double-precision floating-point version of **tanh**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

tanl

long double tanl(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Angle in radians
Returns	The tangent of *x* if successful; 0 if not
errno	**ERANGE**

Calculates the tangent of *x*. Long double-precision floating-point version of **tan**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

tell

long tell(int *handle* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
Returns	The current position of the file pointer (if any) associated with *handle* if successful; –1L if not
errno	**EBADF**

Gets the current position of any file pointer associated with *handle*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

tempnam

char *tempnam(char ***_dir_, char ***_prefix_ **);**

Include	STDIO.H
dir	Target directory if TMP environment variable not defined
prefix	File-name prefix
Returns	A pointer to the name generated if successful; **NULL** if not

Creates a temporary file in another directory.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

time

time_t time(time_t *_timer_);

Include	TIME.H
timer	Storage location for time
Returns	The time in elapsed seconds

Returns the number of seconds elapsed since 00:00:00 Greenwich mean time (GMT), January 1, 1970, according to the system clock.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tmpfile

FILE *tmpfile(void);

Include	STDIO.H
Returns	A stream pointer if successful; a null pointer if not

Creates a temporary file and returns a stream pointer to that file.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tmpnam

char *tmpnam(char *_string_);

Include	STDIO.H
string	Pointer to temporary name
Returns	A pointer to the name generated if successful; **NULL** if not

Generates a string that can be used as the name of a temporary file.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

toascii

int toascii(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The converted character

Sets all but the low-order 7 bits of *c* to 0, so that the converted value represents a character in the ASCII character set. If *c* already represents an ASCII character, *c* is unchanged.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tolower

int tolower(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The lowercase character corresponding to *c*

Converts *c* to lowercase if *c* represents an uppercase letter. Otherwise, *c* is unchanged.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_tolower

int _tolower(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The lowercase character corresponding to *c*

Converts *c* to lowercase. To be used only if *c* is known to be an uppercase letter. The result is undefined if *c* is not an uppercase letter.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

toupper

int toupper(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The uppercase character corresponding to *c*

Converts *c* to uppercase if *c* represents a lowercase letter. Otherwise, *c* is unchanged.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_toupper

int _toupper(int *c* **);**

Include	CTYPE.H
c	Character to be converted
Returns	The uppercase character corresponding to *c*

Converts *c* to uppercase. To be used only if *c* is known to be a lowercase letter. The result is undefined if *c* is not a lowercase letter.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

tzset

void tzset(void);

Include	TIME.H
Returns	No return value

Using the current setting of the environment variable TZ, assigns values to the global variables **daylight**, **timezone**, and **tzname**, which are used by the **ftime** and **localtime** functions to make corrections from GMT to local time, and by **time** to compute GMT from system time.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ultoa

char *ultoa(unsigned long *value*, **char** *****_string_, **int** *radix* **);**

Include	STDLIB.H
value	Number to be converted
string	String result
radix	Base of *value*
Returns	A pointer to *string*

Converts the digits of *value* to a null-terminated character string and stores the result (up to 33 bytes) in *string*.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

umask

int umask(int *pmode* **);**

Include	SYS\TYPES.H, SYS\STAT.H, IO.H
pmode	**S_IWRITE, S_IREAD**
Returns	The previous value of *pmode*

Sets the file-permission mask of the current process to the mode specified by *pmode*. The file-permission mask is used to modify the permission setting of new files created by **creat**, **open**, or **sopen**.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ungetc

int ungetc(int *c*, **FILE ***_stream_ **);**

Include	STDIO.H
c	Character to be pushed
stream	Pointer to target stream
Returns	The character argument *c* if successful; **EOF** if not

Pushes the character *c* back onto the input *stream* and clears the end-of-file indicator. A subsequent read operation on the stream begins with *c*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

ungetch

int ungetch(int *c* **);**

Include	CONIO.H
c	Character to be pushed
Returns	The character argument *c* if successful; **EOF** if not

Pushes the character *c* back to the console, causing *c* to be the next character read by **getch** or **getche**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

unlink

int unlink(const char ***_filename_ **);

Include	IO.H or STDIO.H, ERRNO.H
filename	Path name of file to be removed
Returns	0 if successful; –1 if not
errno	**EACCESS, ENOENT**

Deletes the file specified by *filename*.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_unregisterfonts

void _far _unregisterfonts(void);

Include	GRAPH.H
Returns	No return value

Frees memory previously allocated and used by the **_registerfonts** function. The **_unregisterfonts** function removes the header information for all fonts and unloads the currently selected font data from memory.

☐ ANSI ■ DOS ☐ OS/2 ☐ UNIX ☐ XENIX

utime

int utime(char *_filename_, struct utimbuf *_times_);

Include	SYS\TYPES.H, SYS\UTIME.H, ERRNO.H
filename	File path name
times	Pointer to stored time values
Returns	0 if successful; –1 if not
errno	**EACCESS, EINVAL, EMFILE, ENOENT**

Sets the modification time for the file specified by *filename*.

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

va_arg

type **va_arg(va_list** *arg_ptr*, *type* **);**

Include	STDARG.H (ANSI) or VARARGS.H (UNIX)
type	Type of argument to be retrieved
arg_ptr	Pointer to list of arguments
va_alist	Name of parameter to called function (UNIX version only)
va_dcl	Declaration of **va_alist** (UNIX version only)
Returns	The current argument

Retrieves a value of *type* from the location given by *arg_ptr*. Note that **va_list** is defined as the type of **va_alist**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

va_end

void va_end(va_list *arg_ptr* **);**

Include	STDARG.H or VARARGS.H
arg_ptr	Pointer to list of arguments
va_alist	Name of parameter to called function (UNIX version only)
va_dcl	Declaration of **va_alist** (UNIX version only)
Returns	No return value

Resets *arg_ptr* to **NULL**. Note that **va_list** is defined as the type of **va_alist**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

va_start

void va_start(va_list *arg_ptr* **);** (UNIX version)

void va_start(va_list *arg_ptr*, *prev_param* **);** (ANSI version)

Include	STDARG.H or VARARGS.H
arg_ptr	Pointer to list of arguments
prev_param	Parameter preceding first optional argument (ANSI version only)
va_alist	Name of parameter to called function (UNIX version only)
va_dcl	Declaration of **va_alist** (UNIX version only)
Returns	No return value

Sets *arg_ptr* to beginning of list of optional arguments passed to the function used before **va_arg**. Note that **va_list** is defined as the type of **va_alist**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

vfprintf

int vfprintf(FILE ***_stream_, const char ***_format_**, va_list** *argptr* **);**

Include	STDARG.H or VARARGS.H, STDIO.H
stream	Pointer to target stream
format	Format control string
argptr	Pointer to list of arguments
Returns	The number of characters written if successful; a negative value if not

Formats and outputs data to *stream*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

vprintf

int vprintf(const char ****format*, **va_list *argptr* **);**

Include	STDARG.H or VARARGS.H, STDIO.H
format	Format control string
argptr	Pointer to list of arguments
Returns	The number of characters written, not counting the terminating null character ('\0'), if successful; a negative value if not

Formats and outputs data to the standard output stream, **stdout**.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

vsprintf

int vsprintf(char ****buffer*, **const char ****format*, **va_list *argptr* **);**

Include	STDARG.H or VARARGS.H, STDIO.H
buffer	Storage location for output
format	Format control string
argptr	Pointer to list of arguments
Returns	The number of characters written, not counting the terminating null character ('\0'), if successful; a negative value if not

Formats and outputs data to *buffer*.

■ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

wait

int wait(int ***termstat*** **);**

Include	PROCESS.H, ERRNO.H
termstat	Child process termination status and return code buffer
Returns	The child process identification if returning after normal child process termination; –1 if returning after abnormal child process termination; or –1 immediately otherwise
errno	**ECHILD, EINTR**

Suspends the calling process until any of the caller's immediate child processes terminate. If all of the caller's children have terminated before it calls the **wait** function, the function returns immediately.

☐ ANSI ☐ DOS ■ OS/2 ■ UNIX ■ XENIX

_wrapon

short _far _wrapon(short ***option*** **);**

Include	GRAPH.H
option	**_GWRAPOFF, _GWRAPON**
Returns	The previous value of *option*

Controls whether text output with the **_outtext** and **_outmem** functions wraps to a new line or is simply truncated when the text output reaches the edge of the current text window.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

write

int write(int *handle*, **void *** *buffer*, **unsigned int** *count* **);**

Include	IO.H, ERRNO.H
handle	Handle referring to open file
buffer	Data to be written
count	Number of bytes
Returns	The number of bytes actually written if successful; –1 if not
errno	**EBADF, ENOSPC**

Writes *count* bytes from *buffer* into the file associated with *handle*. The write operation begins at the current position in the file. If the file is open for appending, the operation begins at the current end of the file.

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

yn

double yn(int *n*, **double** *x* **);**

Include	MATH.H, ERRNO.H
n	Integer order
x	Floating-point value
Returns	The result of a Bessel function of *x*; –**HUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order *n*).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_ynl

long double _ynl(int *n*, long double *x*);

Include	MATH.H, ERRNO.H
n	Integer order
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*; **–_LHUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order *n*). Long double-precision floating-point version of **yn**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

y1

double y1(double *x*);

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	The result of a Bessel function of *x*; **–HUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order 1).

☐ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_y1l

long double _y1l(long double *x* **);**

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*; **–_LHUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order 1). Long double-precision floating-point version of **y1**.

□ ANSI ■ DOS ■ OS/2 □ UNIX □ XENIX

y0

double y0(double *x* **);**

Include	MATH.H, ERRNO.H
x	Floating-point value
Returns	The result of a Bessel function of *x*; **–HUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order 0).

□ ANSI ■ DOS ■ OS/2 ■ UNIX ■ XENIX

_y0l

long double _y0l(long double *x*);

Include	MATH.H, ERRNO.H
x	Long double-precision floating-point value
Returns	The result of a Bessel function of *x*; –**_LHUGE_VAL** if *x* is negative
errno	**EDOM**

Returns the Bessel function of the second kind (order 0). Long double-precision floating-point version of **y0**.

☐ ANSI ■ DOS ■ OS/2 ☐ UNIX ☐ XENIX

SCO®
OPEN SYSTEMS SOFTWARE

Please help us to write computer manuals that meet your needs by completing this form. Please post the completed form to the Publications Manager nearest you: The Santa Cruz Operation, Ltd., Croxley Centre, Hatters Lane, Watford WD1 8YN, United Kingdom; The Santa Cruz Operation, Inc., 400 Encinal Street, P.O. Box 1900, Santa Cruz, California 95061, USA or SCO Canada, Inc., 130 Bloor Street West, 10th Floor, Toronto, Ontario, Canada M5S 1N5.

Volume title: _____
(Copy this from the title page of the manual)

Product: _____
(for example, SCO UNIX System V Release 3.2 Operating System Version 4.0)

How long have you used this product?

❑ Less than one month ❑ Less than six months ❑ Less than one year

❑ 1 to 2 years ❑ More than 2 years

How much have you read of this manual?

❑ Entire manual ❑ Specific chapters ❑ Used only for reference

	Agree			*Disagree*	
The software was fully and accurately described	❑	❑	❑	❑	❑
The manual was well organized	❑	❑	❑	❑	❑
The writing was at an appropriate technical level (neither too complicated nor too simple)	❑	❑	❑	❑	❑
It was easy to find the information I was looking for	❑	❑	❑	❑	❑
Examples were clear and easy to follow	❑	❑	❑	❑	❑
Illustrations added to my understanding of the software	❑	❑	❑	❑	❑
I liked the page design of the manual	❑	❑	❑	❑	❑

If you have specific comments or if you have found specific inaccuracies, please report these on the back of this form or on a separate sheet of paper. In the case of inaccuracies, please list the relevant page number.

May we contact you further about how to improve SCO documentation? If so, please supply the following details:

Name _____ *Position* _____

Company _____

Address _____

City & Post/Zip Code _____

Country _____

Telephone _____ *Facsimile* _____