

Compaq C

Run-Time Library Reference Manual for OpenVMS Systems

Order Number: AA-RSMUA-TE

June 2002

This manual describes the functions and macros in the *Compaq C* Run-Time Library for *OpenVMS* systems.

Revision/Update Information: This manual supersedes the *Compaq C Run-Time Library Reference Manual for OpenVMS Systems* for Compaq C Version 6.5.

Software Version: OpenVMS Alpha Version 7.3-1

Compaq Computer Corporation
Houston, Texas

First Printing, February 1991
Revised, June 2002

© 2002 Compaq Information Technologies Group, L.P.

Compaq, the Compaq logo, VAX, Alpha, VMS, OpenVMS, and Tru64 are trademarks of Compaq Information Technologies Group, L.P. in the U.S. and/or other countries. Microsoft is a trademark of Microsoft Corporation in the U.S. and/or other countries. Motif and UNIX are trademarks of The Open Group in the United States and other countries. All other product names mentioned herein may be trademarks of their respective companies.

Confidential computer software. Valid license from Compaq required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is provided "as is" without warranty of any kind and is subject to change without notice. The warranties for Compaq products are set forth in the express limited warranty statements accompanying such products. Nothing herein should be construed as constituting an additional warranty.

ZK5763

The Compaq *OpenVMS* documentation set is available on CD-ROM.

This document was prepared using DECdocument, Version 3.3-1b.

Portions of the *Compaq C* Run-Time Library have been implemented using source copyrighted by the University of California, Berkley and its contributors.

Copyright (c) 1981 Regents of the University of California.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

Preface	xxv
1 Introduction	
1.1 Using the <i>Compaq C</i> Run-Time Library	1-2
1.2 RTL Linking Options on Alpha Systems (<i>Alpha only</i>)	1-3
1.2.1 Linking with the Shareable Image	1-3
1.2.2 Linking with the Object Libraries	1-4
1.2.3 Examples	1-5
1.3 RTL Linking Options on VAX Systems (<i>VAX only</i>)	1-6
1.3.1 Linking with the <i>Compaq C</i> RTL	1-7
1.3.1.1 Linking with the <i>Compaq C</i> RTL Shareable Images	1-7
1.3.1.2 Linking with or Providing Your Own Shareable Images	1-8
1.3.1.3 Linking with the <i>Compaq C</i> RTL Object Libraries	1-8
1.3.1.4 Linking with the <i>Compaq C</i> RTL Object Libraries /NOSYSSHR	1-9
1.3.2 Resolving Link-Time Conflicts with Multiple C RTLs	1-9
1.3.2.1 Using VAXC\$LCL.OPT	1-10
1.3.2.2 Using VAXC\$EMPTY.EXE	1-11
1.3.2.3 Using DECC\$EMPTY.EXE	1-12
1.3.3 Linking Examples for <i>Compaq C</i> or <i>Compaq C++</i> Code Only	1-12
1.3.4 Linking Examples for VAX C and <i>Compaq C</i> Code Combined	1-13
1.3.5 Linking with the VAX C RTL /NOSYSSHR	1-14
1.4 <i>Compaq C</i> RTL Function Prototypes and Syntax	1-14
1.4.1 Function Prototypes	1-14
1.4.2 Syntax Conventions for Function Prototypes	1-15
1.4.3 UNIX Style File Specifications	1-16
1.4.4 Extended File Specifications	1-18
1.5 Feature-Test Macros for Header-File Control	1-18
1.5.1 Standards Macros	1-19
1.5.2 Selecting a Standard	1-19
1.5.3 Interactions with the /STANDARD Qualifier	1-20
1.5.4 Multiple-Version-Support Macro	1-22
1.5.5 Compatibility Modes	1-23
1.5.6 Curses and Socket Compatibility Macros	1-24
1.5.7 2-Gigabyte File Size Macro	1-25
1.5.8 32-bit UID and GID Macros	1-25
1.6 Enabling <i>Compaq C</i> RTL Features Using Feature Logical Names	1-25
1.7 32-Bit User and Group Identifiers	1-35
1.8 Input and Output on <i>OpenVMS</i> Systems	1-35
1.8.1 RMS Record and File Formats	1-37

1.8.2	Access to RMS Files	1-38
1.8.2.1	Accessing RMS Files in Stream Mode	1-39
1.8.2.2	Accessing RMS Record Files in Record Mode	1-39
1.8.2.2.1	Accessing Variable-Length or VFC Record Files in Record Mode	1-41
1.8.2.2.2	Accessing Fixed-Length Record Files in Record Mode	1-42
1.8.2.3	Example—Difference Between Stream Mode and Record Mode . . .	1-42
1.9	Specific Portability Concerns	1-44
1.9.1	Reentrancy	1-46
1.9.2	Multithread Restrictions	1-48
1.10	64-bit Pointer Support (<i>Alpha only</i>)	1-49
1.10.1	Using the <i>Compaq C</i> Run-Time Library	1-49
1.10.2	Obtaining 64-bit Pointers to Memory	1-50
1.10.3	<i>Compaq C</i> Header Files	1-50
1.10.4	Functions Affected	1-51
1.10.4.1	No Pointer-Size Impact	1-51
1.10.4.2	Functions Accepting Both Pointer Sizes	1-52
1.10.4.3	Functions with Two Implementations	1-52
1.10.4.4	Functions Restricted to 32-Bit Pointers	1-53
1.10.5	Reading Header Files	1-54

2 Understanding Input and Output

2.1	Using RMS from RTL Routines	2-4
2.2	UNIX I/O and Standard I/O	2-5
2.3	Wide-Character Versus Byte I/O Functions	2-6
2.4	Conversion Specifications	2-7
2.4.1	Converting Input Information	2-7
2.4.2	Converting Output Information	2-13
2.5	Terminal I/O	2-19
2.6	Program Examples	2-20

3 Character, String, and Argument-List Functions

3.1	Character-Classification Functions	3-4
3.2	Character-Conversion Functions	3-7
3.3	String and Argument-List Functions	3-9
3.4	Program Examples	3-10

4 Error and Signal Handling

4.1	Error Handling	4-2
4.2	Signal Handling	4-5
4.2.1	<i>OpenVMS</i> Versus UNIX Terminology	4-5
4.2.2	UNIX Signals and the <i>Compaq C</i> RTL	4-6
4.2.3	Signal-Handling Concepts	4-7
4.2.4	Signal Actions	4-8
4.2.5	Signal Handling and <i>OpenVMS</i> Exception Handling	4-9
4.3	Program Example	4-13

5 Subprocess Functions

5.1	Implementing Child Processes in <i>Compaq C</i>	5-2
5.2	The <code>exec</code> Functions	5-3
5.2.1	<code>exec</code> Processing	5-4
5.2.2	<code>exec</code> Error Conditions	5-4
5.3	Synchronizing Processes	5-5
5.4	Interprocess Communication	5-5
5.5	Program Examples	5-5

6 Curses Screen Management Functions and Macros

6.1	Using the BSD-Based Curses Package (<i>Alpha only</i>)	6-1
6.2	Curses Overview	6-2
6.3	Curses Terminology	6-4
6.3.1	Predefined Windows (<code>stdscr</code> and <code>curscr</code>)	6-5
6.3.2	User-Defined Windows	6-5
6.4	Getting Started with Curses	6-7
6.5	Predefined Variables and Constants	6-9
6.6	Cursor Movement	6-10
6.7	Program Example	6-11

7 Math Functions

7.1	Math Function Variants—float, long double	7-3
7.2	Error Detection	7-4
7.3	The <code><fp.h></code> Header File	7-4
7.4	Example	7-5

8 Memory Allocation Functions

8.1	Program Example	8-2

9 System Functions

10 Developing International Software

10.1	Internationalization Support	10-1
10.1.1	Installation	10-1
10.1.2	Unicode Support	10-1
10.2	Features of International Software	10-2
10.3	Developing International Software Using <i>Compaq C</i>	10-3
10.4	Locales	10-3
10.5	Using the <code>setlocale</code> Function to Set Up an International Environment	10-4
10.6	Using Message Catalogs	10-5
10.7	Handling Different Character Sets	10-6
10.7.1	Charmap File	10-6
10.7.2	Converter Functions	10-6
10.7.3	Using Codeset Converter Files	10-6
10.8	Handling Culture-Specific Information	10-7
10.8.1	Extracting Cultural Information From a Locale	10-8
10.8.2	Date and Time Formatting Functions	10-8
10.8.3	Monetary Formatting Function	10-8
10.8.4	Numeric Formatting	10-8

10.9	Functions for Handling Wide Characters	10-9
10.9.1	Character Classification Functions	10-9
10.9.2	Case Conversion Functions	10-9
10.9.3	Functions for Input and Output of Wide Characters	10-10
10.9.4	Functions for Converting Multibyte and Wide Characters	10-10
10.9.5	Functions for Manipulating Wide-Character Strings and Arrays	10-11
10.10	Collating Functions	10-11

11 Date/Time Functions

11.1	Date/Time Support Models	11-1
11.2	Overview of Date/Time Functions	11-2
11.3	<i>Compaq C</i> RTL Date/Time Computations—UTC and Local Time	11-3
11.4	Time-Zone Conversion Rule Files	11-4
11.5	Sample Date/Time Scenario	11-5

Reference Section

abort	REF-3
abs	REF-4
access	REF-5
acos	REF-7
acosh (<i>Alpha only</i>)	REF-8
[w]addch	REF-9
[w]addstr	REF-10
alarm	REF-11
asctime, asctime_r	REF-12
asin	REF-14
asinh (<i>Alpha only</i>)	REF-15
assert	REF-16
atan	REF-17
atan2	REF-18
atanh (<i>Alpha only</i>)	REF-19
atexit	REF-20
atof	REF-21
atoi, atol	REF-22
atoq, atoll (<i>Alpha only</i>)	REF-23
basename	REF-24
bcmp	REF-25
bcopy	REF-26
box	REF-27
brk	REF-28
bsearch	REF-29
btowc	REF-31
bzero	REF-32
cabs	REF-33
calloc	REF-34
catclose	REF-35
catgets	REF-36

catopen	REF-39
cbrt (<i>Alpha only</i>)	REF-42
ceil	REF-43
cfree	REF-44
chdir	REF-45
chmod	REF-46
chown	REF-47
[w]clear	REF-48
clearerr	REF-49
clearok	REF-50
clock	REF-51
close	REF-52
closedir	REF-53
[w]clrattr	REF-55
[w]clrto bot	REF-56
[w]clrtoeol	REF-57
confstr	REF-58
copysign (<i>Alpha only</i>)	REF-60
cos	REF-61
cosh	REF-62
cot	REF-63
creat	REF-64
[no]crmode	REF-70
ctermid	REF-71
ctime, ctime_r	REF-72
cuserid	REF-74
DECC\$CRTL_INIT	REF-75
decc\$feature_get_index	REF-76
decc\$feature_get_name	REF-78
decc\$feature_get_value	REF-79
decc\$feature_set_value	REF-80
decc\$fix_time	REF-81
decc\$from_vms	REF-82
decc\$match_wild	REF-84
decc\$record_read	REF-85
decc\$record_write	REF-86
decc\$set_child_standard_streams	REF-87
decc\$set_reentrancy	REF-91
decc\$to_vms	REF-93
decc\$translate_vms	REF-95
decc\$validate_wchar	REF-97
decc\$write_eof_to_mbx	REF-98
[w]delch	REF-101
delete	REF-102
[w]deleteln	REF-103
delwin	REF-104
difftime	REF-105

dirname	REF-106
div	REF-108
dlclose	REF-109
dLError	REF-110
dlopen	REF-111
dlsym	REF-112
drand48	REF-113
dup, dup2	REF-114
[no]echo	REF-115
ecvt	REF-116
endpwent	REF-118
endwin	REF-119
erand48	REF-120
[w]erase	REF-121
erf	REF-122
execl	REF-123
execle	REF-125
execlp	REF-126
execv	REF-127
execve	REF-128
execvp	REF-129
exit, _exit	REF-130
exp	REF-131
fabs	REF-132
fchown	REF-133
fclose	REF-134
fcntl	REF-135
fcvt	REF-138
fdopen	REF-140
feof	REF-141
ferror	REF-142
fflush	REF-143
ffs	REF-144
fgetc	REF-145
fgetname	REF-146
fgetpos	REF-147
fgets	REF-149
fgetwc	REF-151
fgetws	REF-152
fileno	REF-154
finite (<i>Alpha only</i>)	REF-155
floor	REF-156
fmod	REF-157
fopen	REF-158
fp_class (<i>Alpha only</i>)	REF-160
fpathconf	REF-161
fprintf	REF-163

fputc	REF-165
fputs	REF-166
fputwc	REF-167
fputws	REF-168
fread	REF-169
free	REF-170
freopen	REF-171
frexp	REF-172
fscanf	REF-174
fseek	REF-176
fseeko	REF-178
fsetpos	REF-179
fstat	REF-180
fsync	REF-183
ftell	REF-184
ftello	REF-185
ftime	REF-186
ftruncate	REF-187
ftw	REF-188
fwait	REF-190
fwide	REF-191
fwprintf	REF-192
fwrite	REF-194
fwscanf	REF-195
gcv	REF-197
getc	REF-199
[w]getch	REF-200
getchar	REF-201
getclock	REF-202
getcwd	REF-203
getdtablesize	REF-204
getegid	REF-205
getenv	REF-206
geteuid	REF-208
getgid	REF-209
getitimer	REF-210
getlogin	REF-212
getname	REF-213
getopt	REF-214
getpagesize	REF-217
getpid	REF-218
getppid	REF-219
getpwent	REF-220
getpwnam	REF-221
getpwuid	REF-222
gets	REF-223
[w]getstr	REF-224

gettimeofday	REF-225
getuid	REF-226
getw	REF-227
getwc	REF-228
getwchar	REF-229
getyx	REF-230
gmtime, gmtime_r	REF-231
gsignal	REF-233
hypot	REF-235
iconv	REF-236
iconv_close	REF-238
iconv_open	REF-239
[w]inch	REF-241
index	REF-242
initscr	REF-243
initstate	REF-244
[w]insch	REF-246
[w]insertln	REF-247
[w]insstr	REF-248
isalnum	REF-249
isalpha	REF-250
isapipe	REF-251
isascii	REF-252
isatty	REF-253
isctrl	REF-254
isdigit	REF-255
isgraph	REF-256
islower	REF-257
isnan (<i>Alpha only</i>)	REF-258
isprint	REF-259
ispunct	REF-260
isspace	REF-261
isupper	REF-262
iswalnum	REF-263
iswalpha	REF-264
iswctrl	REF-265
iswctype	REF-266
iswdigit	REF-268
iswgraph	REF-269
iswlower	REF-270
iswprint	REF-271
iswpunct	REF-272
iswspace	REF-273
iswupper	REF-274
iswxdigit	REF-275
isxdigit	REF-276
j0, j1, jn (<i>Alpha only</i>)	REF-277

jrand48	REF-278
kill	REF-279
labs	REF-280
lcong48	REF-281
ldexp	REF-282
ldiv	REF-283
leaveok	REF-284
lgamma (<i>Alpha only</i>)	REF-285
link	REF-286
localeconv	REF-287
localtime, localtime_r	REF-291
log, log2, log10	REF-293
log1p (<i>Alpha only</i>)	REF-294
logb (<i>Alpha only</i>)	REF-295
longjmp	REF-296
longname	REF-298
lrnd48	REF-299
lseek	REF-300
lwait	REF-302
malloc	REF-303
mblen	REF-305
mbrlen	REF-306
mbrtowc	REF-307
mbstowcs	REF-309
mbtowc	REF-310
mbsinit	REF-311
mbsrtowcs	REF-312
memccpy	REF-314
memchr	REF-315
memcmp	REF-316
memcpy	REF-317
memmove	REF-318
memset	REF-319
mkdir	REF-320
mkstemp	REF-323
mktemp	REF-324
mktime	REF-325
mmap	REF-326
modf	REF-331
[w]move	REF-332
mprotect	REF-333
mrnd48	REF-335
msync	REF-336
munmap	REF-338
mv[w]addch	REF-339
mv[w]addstr	REF-340
mvcur	REF-341

mv[w]delch	REF-342
mv[w]getch	REF-343
mv[w]getstr	REF-344
mv[w]inch	REF-345
mv[w]insch	REF-346
mv[w]insstr	REF-347
mvwin	REF-348
newwin	REF-349
nextafter (<i>Alpha only</i>)	REF-350
nice	REF-351
nint (<i>Alpha only</i>)	REF-352
[no]nl	REF-353
nl_langinfo	REF-354
nrnd48	REF-358
open	REF-359
opendir	REF-362
overlay	REF-364
overwrite	REF-365
pathconf	REF-366
pause	REF-368
pclose	REF-369
perror	REF-370
pipe	REF-371
popen	REF-374
pow	REF-375
printf	REF-376
[w]printw	REF-377
putc	REF-378
putchar	REF-379
putenv	REF-380
puts	REF-381
putw	REF-382
putwc	REF-383
putwchar	REF-384
qabs, llabs (<i>Alpha only</i>)	REF-385
qdiv, lldiv (<i>Alpha only</i>)	REF-386
qsort	REF-387
raise	REF-388
rand	REF-389
random	REF-390
[no]raw	REF-391
read	REF-393
readdir, readdir_r	REF-395
realloc	REF-397
[w]refresh	REF-398
remove	REF-399
rename	REF-400

rewind	REF-401
rewinddir	REF-402
rindex	REF-403
rint (<i>Alpha only</i>)	REF-404
rmdir	REF-405
sbrk	REF-406
scalb (<i>Alpha only</i>)	REF-407
scanf	REF-408
[w]scanw	REF-409
scroll	REF-410
scrollok	REF-411
seed48	REF-412
seekdir	REF-413
[w]setattr	REF-414
setbuf	REF-415
setenv	REF-416
setgid	REF-417
setitimer	REF-419
setjmp	REF-421
setlocale	REF-423
setpwent	REF-427
setstate	REF-428
setuid	REF-429
setvbuf	REF-431
sigaction	REF-433
sigaddset	REF-436
sigblock	REF-437
sigdelset	REF-438
sigemptyset	REF-439
sigfillset	REF-440
sigismember	REF-441
siglongjmp	REF-442
sigmask	REF-443
signal	REF-444
sigpause	REF-445
sigpending	REF-446
sigprocmask	REF-447
sigsetjmp	REF-449
sigsetmask	REF-451
sigstack (<i>VAX only</i>)	REF-452
sigsuspend	REF-454
sigvec	REF-455
sin	REF-456
sinh	REF-457
sleep	REF-458
sprintf	REF-459
sqrt	REF-461

srand	REF-462
srand48	REF-463
srandom	REF-464
sscanf	REF-465
ssignal	REF-467
[w]standend	REF-468
[w]standout	REF-469
stat	REF-470
strcasecmp	REF-473
strcat	REF-474
strchr	REF-476
strcmp	REF-478
strcoll	REF-479
strcpy	REF-480
strcspn	REF-481
strdup	REF-482
strerror	REF-483
strfmon	REF-485
strftime	REF-489
strlen	REF-495
strncasecmp	REF-496
strncat	REF-497
strncmp	REF-498
strncpy	REF-500
strnlen	REF-501
strpbrk	REF-502
strptime	REF-503
strrchr	REF-508
strsep	REF-509
strspn	REF-510
strstr	REF-511
strtod	REF-513
strtok	REF-515
strtol	REF-517
strtod, strtoll (<i>Alpha only</i>)	REF-519
strtoul	REF-521
strtouq, strtoull (<i>Alpha only</i>)	REF-522
strxfrm	REF-523
subwin	REF-526
swab	REF-527
swprintf	REF-528
swscanf	REF-529
sysconf	REF-530
system	REF-536
tan	REF-538
tanh	REF-539
telldir	REF-540

tempnam	REF-541
time	REF-542
times	REF-543
tmpfile	REF-544
tmpnam	REF-545
toascii	REF-546
tolower	REF-547
_tolower	REF-548
touchwin	REF-549
toupper	REF-550
_toupper	REF-551
towctrans	REF-552
towlower	REF-553
towupper	REF-554
trunc (<i>Alpha only</i>)	REF-555
truncate	REF-556
ttyname	REF-557
tzset	REF-558
ualarm	REF-562
umask	REF-563
uname	REF-564
ungetc	REF-565
ungetwc	REF-566
unordered (<i>Alpha only</i>)	REF-567
utime	REF-568
utimes	REF-571
unsetenv	REF-574
usleep	REF-575
VAXC\$CRTL_INIT	REF-576
VAXC\$ESTABLISH	REF-577
va_arg	REF-579
va_count	REF-580
va_end	REF-581
va_start_1, va_start	REF-582
vfork	REF-584
vfprintf	REF-586
vfscanf	REF-587
vfwprintf	REF-589
vfwscanf	REF-591
vprintf	REF-592
vscanf	REF-593
vsprintf	REF-594
vsscanf	REF-595
vswprintf	REF-596
vswscanf	REF-598
vwprintf	REF-599
vwscanf	REF-600

wait	REF-601
wait3	REF-602
wait4	REF-605
waitpid	REF-608
wertomb	REF-612
wscat	REF-613
wcschr	REF-615
wscmp	REF-617
wscoll	REF-618
wscpy	REF-619
wscspn	REF-620
wcsftime	REF-622
wcslen	REF-628
wcsncat	REF-629
wcsncmp	REF-631
wcsncpy	REF-632
wcspbrk	REF-633
wcsrchr	REF-634
wcsrtombs	REF-636
wcsspn	REF-638
wcsstr	REF-640
wctod	REF-641
wctok	REF-643
wctol	REF-646
wctombs	REF-648
wctoul	REF-649
wcswcs	REF-652
wcswidth	REF-654
wcsxfrm	REF-655
wctob	REF-658
wctomb	REF-659
wctrans	REF-660
wctype	REF-661
wcwidth	REF-664
wmemchr	REF-665
wmemcmp	REF-666
wmemcpy	REF-667
wmemmove	REF-668
wmemset	REF-669
wprintf	REF-670
wrapok	REF-672
write	REF-673
writew	REF-674
wscanf	REF-676
y0, y1, yn (<i>Alpha only</i>)	REF-677

A *Compaq C* Socket Routine Reference

A.1	Porting Considerations	A-1
A.1.1	Calling an IPC Routine from an AST State	A-1
A.1.2	Calling from KERNEL or EXEC Modes	A-1
A.1.3	Event Flags	A-2
A.1.4	Suppressing <i>Compaq C</i> Compilation Warnings	A-2
A.1.5	Header Files	A-2
A.2	<i>Compaq C</i> Structures	A-2
A.3	Internet Protocols	A-2
A.3.1	Transmission Control Protocol	A-3
A.3.2	User Datagram Protocol	A-3
A.4	errno Values	A-4
A.5	h_errno Values	A-6
A.6	Relationship Between errno and h_errno	A-7
A.7	TCP/IP Interface Enhancements	A-8
A.8	Summary of Socket Routines	A-9
A.8.1	Basic Communication Routines	A-9
A.8.2	Auxiliary Communication Routines	A-10
A.8.3	h_errno Support Routines	A-10
A.8.4	Communication Support Routines	A-10
A.8.5	IPv6 Routines	A-12
	accept	A-13
	bind	A-15
	close	A-17
	connect	A-18
	decc\$get_sdc	A-20
	endhostent	A-21
	endnetent	A-22
	endprotoent	A-23
	endservent	A-24
	gethostaddr	A-25
	gethostbyaddr	A-26
	gethostbyname	A-27
	gethostent	A-28
	gethostname	A-29
	getnetbyaddr	A-30
	getnetbyname	A-31
	getnetent	A-32
	getpeername	A-33
	getprotobyname	A-34
	getprotobynumber	A-35
	getprotoent	A-36
	getservbyname	A-37
	getservbyport	A-38
	getservent	A-39
	getsockname	A-40
	getsockopt	A-41
	herror	A-43
	hstrerror	A-44

hostalias	A-45
htonl	A-46
htons	A-47
inet_addr	A-48
inet_lnaof	A-49
inet_makeaddr	A-50
inet_netof	A-51
inet_network	A-52
inet_ntoa	A-53
ioctl	A-54
listen	A-55
ntohl	A-56
ntohs	A-57
read	A-58
recv	A-59
recvfrom	A-61
recvmsg	A-63
select	A-65
send	A-68
sendmsg	A-70
sendto	A-72
sethostent	A-74
setnetent	A-75
setprotoent	A-76
setservent	A-77
setsockopt	A-78
shutdown	A-80
socket	A-81
decc\$socket_fd	A-83
vaxc\$get_sdc	A-84
write	A-85
A.9 Programming Examples	A-86

B Version-Dependency Tables

B.1	Functions Available on all <i>OpenVMS VAX</i> and <i>OpenVMS Alpha</i> Versions	B-1
B.2	Functions Available on <i>OpenVMS</i> Version 6.2 and Higher	B-3
B.3	Functions Available on <i>OpenVMS</i> Version 7.0 and Higher	B-4
B.4	Functions Available on <i>OpenVMS Alpha</i> Version 7.0 and Higher	B-5
B.5	Functions Available on <i>OpenVMS</i> Version 7.2 and Higher	B-6
B.6	Functions Available on <i>OpenVMS</i> Version 7.3 and Higher	B-6
B.7	Functions Available on <i>OpenVMS</i> Version 7.3-1 and Higher	B-6

C Prototypes Duplicated to Non-Standard Headers

Index

Examples

1-1	Differences Between Stream Mode and Record Mode Access	1-42
2-1	Output of the Conversion Specifications	2-20
2-2	Using the Standard I/O Functions	2-22
2-3	Using Wide Character I/O Functions	2-23
2-4	I/O Using File Descriptors and Pointers	2-24
3-1	Character-Classification Functions	3-7
3-2	Converting Double Values to an ASCII String	3-8
3-3	Changing Characters to and from Uppercase Letters	3-8
3-4	Concatenating Two Strings	3-10
3-5	Four Arguments to the <code>strncpy</code> Function	3-10
3-6	Using the <code><stdarg.h></code> Functions and Definitions	3-11
4-1	Suspending and Resuming Programs	4-13
5-1	Creating the Child Process	5-5
5-2	Passing Arguments to the Child Process	5-7
5-3	Checking the Status of Child Processes	5-8
5-4	Communicating Through a Pipe	5-9
6-1	A Curses Program	6-7
6-2	Manipulating Windows	6-8
6-3	Refreshing the Terminal Screen	6-9
6-4	Curses Predefined Variables	6-10
6-5	The Cursor Movement Functions	6-11
6-6	<code>stdscr</code> and Occluding Windows	6-11
7-1	Calculating and Verifying a Tangent Value	7-5
8-1	Allocating and Deallocating Memory for Structures	8-2
9-1	Accessing the User Name	9-3
9-2	Accessing Terminal Information	9-4
9-3	Manipulating the Default Directory	9-4
9-4	Printing the Date and Time	9-4
A-1	TCP/IP Server	A-86
A-2	TCP/IP Client	A-91
A-3	UDP/IP Server	A-96
A-4	UDP/IP Client	A-102

Figures

1-1	Linking with the <i>Compaq C</i> RTL on <i>OpenVMS Alpha</i> Systems	1-6
1-2	I/O Interface from C Programs	1-36
1-3	Mapping Standard I/O and UNIX I/O to RMS	1-37
5-1	Communications Links Between Parent and Child Processes	5-2
6-1	An Example of the <code>stdscr</code> Window	6-5
6-2	Displaying Windows and Subwindows	6-6

6-3	Updating the Terminal Screen	6-7
6-4	An Example of the getch Macro	6-13
REF-1	Reading and Writing to a Pipe	REF-373

Tables

1-1	Linking Conflicts	1-10
1-2	UNIX and <i>OpenVMS</i> File Specification Delimiters	1-16
1-3	Valid and Invalid UNIX and <i>OpenVMS</i> File Specifications	1-17
1-4	Feature Test Macros - Standards	1-19
1-5	C RTL Feature Logical Names	1-26
1-6	Functions with Dual Implementations	1-53
1-7	Functions restricted to 32-bit pointers	1-53
1-8	Callbacks that Pass Only 32-Bit Pointers	1-54
2-1	I/O Functions and Macros	2-1
2-2	Optional Characters Between % (or %n\$) and the Input Conversion Specifier	2-8
2-3	Conversion Specifiers for Formatted Input	2-9
2-4	Optional Characters Between % (or %n\$) and the Output Conversion Specifier	2-14
2-5	Conversion Specifiers for Formatted Output	2-16
3-1	Character, String, and Argument-List Functions	3-1
3-2	Character-Classification Functions	3-4
3-3	ASCII Characters and the Character-Classification Functions	3-5
4-1	Error- and Signal-Handling Functions	4-1
4-2	The Error Code Symbolic Values	4-2
4-3	<i>Compaq C</i> RTL Signals	4-6
4-4	<i>Compaq C</i> RTL Signals and Corresponding <i>OpenVMS</i> VAX Exceptions (<i>VAX only</i>)	4-10
4-5	<i>Compaq C</i> RTL Signals and Corresponding <i>OpenVMS</i> Alpha Exceptions (<i>Alpha only</i>)	4-11
5-1	Subprocess Functions	5-1
6-1	Curses Functions and Macros	6-2
6-2	Curses Predefined Variables and #define Constants	6-9
7-1	Math Functions	7-1
8-1	Memory Allocation Functions	8-1
9-1	System Functions	9-1
10-1	Locale Categories	10-3
11-1	Date/Time Functions	11-1
11-2	Time-zone Filename Acronyms	11-5
REF-1	Interpretation of the mode Argument	REF-5
REF-2	File Protection Values and Their Meanings	REF-46
REF-3	RMS Valid Keywords and Values	REF-65
REF-4	tm Structure	REF-291
REF-5	Optional Characters in strfmon Conversion Specifications	REF-486
REF-6	strfmon Conversion Specifiers	REF-487
REF-7	Optional Elements of strftime Conversion Specifications	REF-490

REF-8	strftime Conversion Specifiers	REF-490
REF-9	strptime Conversion Specifications	REF-504
REF-10	SYSCONF Argument and Return Values	REF-530
REF-11	Time-Zone Initialization Rules	REF-559
REF-12	The vfork and fork Functions	REF-584
REF-13	Optional Elements of wcsftime Conversion Specifications	REF-623
REF-14	wcsftime Conversion Specifiers	REF-623
A-1	Structures Used by Socket Routines	A-2
A-2	errno Values	A-4
A-3	4.4BSD Entry Points	A-9
A-4	Basic Communication Routines	A-9
A-5	Auxiliary Communication Routines	A-10
A-6	Supported h_errno Routines	A-10
A-7	Supported Communication Routines	A-10
A-8	IPv6 Routines	A-12
B-1	Functions Available on All <i>OpenVMS</i> Systems	B-1
B-2	Functions Added in <i>OpenVMS</i> Version 6.2	B-3
B-3	Functions Added in <i>OpenVMS</i> Version 7.0	B-4
B-4	Functions Added in <i>OpenVMS Alpha</i> Version 7.0	B-5
B-5	Functions Added in <i>OpenVMS</i> Version 7.2	B-6
B-6	Functions Added in <i>OpenVMS</i> Version 7.3	B-6
B-7	Functions Added in <i>OpenVMS</i> Version 7.3-1	B-6
C-1	Duplicated Prototypes	C-1

Preface

This manual describes the *Compaq C* Run-Time Library (RTL). It provides reference information about the RTL functions and macros that perform input/output (I/O) operations, character and string manipulation, mathematical operations, error detection, subprocess creation, system access, and screen management.

It also notes portability concerns between operating systems, where applicable, and describes the *Compaq C* for *OpenVMS* socket routines used for writing Internet application programs for the *TCP/IP Services for OpenVMS* product or other implementations of the TCP/IP protocol.

The *Compaq C* RTL contains XPG4-compliant internationalization support, providing functions to help you develop software that can run in different languages and cultures.

You can send comments or suggestions regarding this manual or any *Compaq C* document by sending electronic mail to the following Internet address:

`c_docs@compaq.com`

Intended Audience

This manual is intended for experienced and novice programmers who need reference information on the functions and macros found in the *Compaq C* RTL.

Document Structure

This manual has the following chapters, reference section, and appendixes:

- Chapter 1 provides an overview of the *Compaq C* RTL.
- Chapter 2 discusses the Standard I/O, Terminal I/O, and UNIX I/O functions.
- Chapter 3 describes the character, string, and argument-list functions.
- Chapter 4 describes the error-handling and signal-handling functions.
- Chapter 5 explains the functions used to create subprocesses.
- Chapter 6 describes the Curses Screen Management functions.
- Chapter 7 discusses the math functions.
- Chapter 8 explains the memory allocation functions.
- Chapter 9 describes the functions used to interact with the operating system.
- Chapter 10 gives an introduction to the facilities provided in the *Compaq C* environment on *OpenVMS* systems for developing international software.
- Chapter 11 describes the date/time functions.
- The Reference Section describes all the functions in the *Compaq C* RTL.

- Appendix A describes the *Compaq C* for *OpenVMS* socket routines used for writing Internet application programs for the *TCP/IP Services for OpenVMS*.
- Appendix B contains version-dependency tables that list the *Compaq C* RTL functions supported on different *OpenVMS* versions.
- Appendix C lists the function prototypes that are duplicated in more than one header file.

Associated Documents

The following documents may be useful when programming in *Compaq C* for *OpenVMS* Systems:

- *Compaq C User's Guide for OpenVMS Systems*—For C programmers who need information on using *Compaq C* for *OpenVMS* Systems.
- *Compaq C Language Reference Manual*—Provides language reference information for *Compaq C* on Compaq systems.
- *VAX C to Compaq C Migration Guide*—To help *OpenVMS* VAX application programmers migrate from VAX C to *Compaq C*.
- *Compaq C Installation Guide for OpenVMS VAX Systems*—For *OpenVMS* system programmers who install the *Compaq C* software on VAX systems.
- *Compaq C Installation Guide for OpenVMS Alpha Systems*—For *OpenVMS* system programmers who install the *Compaq C* software on *Alpha* systems.
- *OpenVMS Master Index*—For programmers who need to work with the VAX and Alpha machine architectures or the *OpenVMS* system services. This index lists manuals that cover the individual topics concerning access to the *OpenVMS* operating system.
- *X/Open Portability Guide, Issue 3*—Documents what is commonly known as the XPG3 specification.
- *X/Open CAE Specification System Interfaces and Headers, Issue 4*—Documents what is commonly known as the XPG4 specification.
- *X/Open CAE Specification, System Interfaces and Headers, Issue 4, Version 2*—Documents what is commonly known as XPG4 V2.
- *Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API)—Amendment 2: Threads Extension [C Language]*—Documents what is also known as POSIX 1003.1c-1995.
- *ISO/IEC 9945-2:1993 - Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities*—Documents what is also known as ISO POSIX-2.
- *ISO/IEC 9945-1:1990 - Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Programming Interface (API) (C Language)*—Documents what is also known as ISO POSIX-1.
- *ANSI/ISO/IEC 9899:1999 - Programming Languages - C*—The C99 standard, published by ISO in December, 1999 and adopted as an ANSI standard in April, 2000.
- *ISO/IEC 9899:1990-1994 - Programming Languages - C, Amendment 1: Integrity*—Documents what is also known as ISO C, Amendment 1.

- *ISO/IEC 9899:1990[1992] - Programming Languages - C*—Documents what is also known as ISO C. The normative part is the same as *X3.159-1989, American National Standard for Information Systems - Programming Language C*, also known as ANSI C.

For additional information about Compaq *OpenVMS* products and services, access the Compaq website at the following location:

<http://www.openvms.compaq.com/>

How to Order Additional Documentation

Visit the following World Wide Web address for information about how to order additional documentation:

<http://www.openvms.compaq.com/>

Conventions Used in this Document

Convention	Meaning
<code>Return</code>	The symbol <code>Return</code> represents a single stroke of the Return key on a terminal.
<code>Ctrl/X</code>	The symbol Ctrl/X, where letter X represents a terminal control character, is generated by holding down the Ctrl key while pressing the key of the specified terminal character.
switch statement int data type fprintf function <stdio.h> header file	Monospace type identifies language keywords and the names of <i>Compaq C</i> functions and header files. Monospace type is also used when referring to a specific variable name used in an example.
<i>arg1</i>	Italic type indicates a placeholder, such as an argument or parameter name, and the introduction of new terms.
\$ RUN CPROG <code>Return</code>	Interactive examples show user input in boldface type.
float x; . . . x = 5; option, . . .	A vertical ellipsis indicates that not all of the text of a program or program output is illustrated. Only relevant material is shown in the example.
[output-source, . . .]	A horizontal ellipsis indicates that additional parameters, options, or values can be entered. A comma that precedes the ellipsis indicates that successive items must be separated by commas.
	Square brackets, in function synopses and a few other contexts, indicate that a syntactic element is optional. Square brackets are not optional, however, when used to delimit a directory name in an <i>OpenVMS</i> file specification or when used to delimit the dimensions of a multidimensional array in <i>Compaq C</i> source code.

Convention	Meaning
sc-specifier ::= auto static extern register	In syntax definitions, items appearing on separate lines are mutually exclusive alternatives.
[a b]	Brackets surrounding two or more items separated by a vertical bar () indicate a choice; you must choose one of the two syntactic elements.
Δ	A delta symbol is used in some contexts to indicate a single ASCII space character.

Platform Labels

A *platform* is a combination of operating system and hardware that provides a distinct environment. This manual contains information applicable to the *OpenVMS* operating system on both the VAX and Alpha architectures.

The information in this manual applies to both of these platforms, except when specifically labeled, as follows:

Label	Explanation
(Alpha only)	Specific to an Alpha processor (Alpha architecture) running the <i>OpenVMS</i> operating system.
(VAX only)	Specific to a VAX processor running the <i>OpenVMS</i> operating system.

New and Changed Features - OpenVMS Version 7.3-1

The following enhancements have been made to the Compaq C Run-Time Library for OpenVMS Version 7.3-1:

- Support for larger than 2-gigabyte files
- Enabling C RTL features at compile/link time
- UNIX style file name enhanced support
- POSIX root support
- 32-bit user and group identifiers
- New and changed C RTL functions

Larger Than 2-Gigabyte Files

Support has been added for compiling applications to use file sizes and offsets that are two gigabytes and larger. This is accomplished by allowing file offsets of 64-bit integers. Two new functions, `fseeko` and `ftello`, have been added. They are identical to `fseek` and `ftell`, but they accept or return values of type `off_t`, which allows for a 64-bit variant of `off_t` to be used.

Modifications to accommodate a 64-bit file offset have been made to the existing C RTL functions `lseek`, `mmap`, `ftuncate`, `truncate`, `stat`, `fstat`, and `ftw`.

The new 64-bit interfaces can be selected at compile time by defining the `_LARGEFILE` feature macro:

```
$ CC /DEFINE=_LARGEFILE
```

Enabling C RTL Features at Compile/Link Time

The C RTL now provides an extensive list of feature switches that affect an application's behavior. They can be used to enable new behaviors or preserve previous, deprecated behaviors. You can enable or disable these features at run time using `DECC$` logical names. See Section 1.6 for a description of these feature switches.

A programmable interface is available to sense or set feature switches within the application. By using this interface in a function referenced by the global symbol `LIB$INITIALIZE`, you can enable or disable features before the C RTL is initialized.

In many instances, you can enable new behaviors by using feature switches. This will preserve existing behavior as the default. If an application needs to select a different default behavior, a user can then modify the run-time behavior using a logical name.

For some applications, some feature-switch settings may be necessary. In this case, they can be set in a way that a logical name cannot override the value.

There are four functions provided to do this:

```
int decc$feature_get_index(char *name);
char *decc$feature_get_name(int index);
int decc$feature_get_value(int index, int mode);
int decc$feature_set_value(int index, int mode, int value);
```

See the reference section of this manual for a description of the `decc$feature` functions.

UNIX Style File Name Enhanced Support

The Compaq C RTL can be instructed to accept UNIX style file and directory names that contain the extended ASCII character set supported by OpenVMS Extended File Specifications (EFS) on ODS-5 devices.

Note

This feature is intended for use by applications ported from UNIX systems, and not for native OpenVMS applications.

To enable this feature, define the `DECC$FILENAME_UNIX_ONLY` logical name to "ENABLE" before program execution:

```
$ DEFINE DECC$FILENAME_UNIX_ONLY ENABLE
```

With this feature enabled, all file and path specifications input to the Compaq C RTL are assumed to be UNIX style, and can consist of the full printable ASCII character set (20-7E hexadecimal), with the following exceptions:

* ? / " (asterisk, question-mark, forward-slash, and double-quotes)

The Compaq C RTL provides any necessary enclosing quotation marks for special characters when presenting the file specification to the OpenVMS file system.

With this feature enabled, OpenVMS file and path specifications that are output by the Compaq C RTL are converted to UNIX style where appropriate, and any special characters quoted by the OpenVMS file system are unquoted before being returned to the Compaq C RTL caller. In addition, the OpenVMS version-number field is stripped from the file name on output.

For example, assume the following UNIX style path name is input:

```
./abc.d/data.dat.dat
```

The Compaq C RTL converts it to the following before passing it on to the OpenVMS file system:

```
[.abc^.d]data^.dat.dat
```

The '^' is the EFS quote or escape character. (For more information about escape characters and name and type delimiters, see the *OpenVMS Guide to Extended File Specifications*.) When this file name is retrieved, it is output to the Compaq C RTL caller just as it was input, with quoted characters unquoted.

Note that under normal Compaq C RTL parsing rules, an OpenVMS version field in a UNIX style input file name is passed directly to the file system. Therefore, the following example accesses version number 3 of the file with name "data" and type "dat":

```
data.dat;3
```

With the new feature, this file name is passed to the OpenVMS file system as the following (a file with name "data" and type "dat;3"):

```
data.dat^;3
```

The OpenVMS version number associated with the file is not returned on output.

The scope of this feature applies to allowable characters in both directory and file-name fields of a UNIX style specification. Enabling this feature does not affect other features of the Compaq C RTL UNIX-to-OpenVMS and OpenVMS-to-UNIX path conversion. If logical-name expansion of the first element in the UNIX specification is enabled, the Compaq C RTL does not quote any special characters appearing in the logical-name equivalence string before presentation to the file system.

See Section 1.4.4 and Section 1.6 for additional information on UNIX related feature logical names.

POSIX Root Support

Support has been added to provide OpenVMS with a UNIX style root that behaves like a real directory. This allows such actions as:

```
% cd /
% mkdir /dirname
% tar -xvf tarfile.tar /dirname
% ls /
```

Previously, the C RTL did not recognize "/" as a directory name. The normal processing for a file path starting with "/" was to interpret the first element as a logical name or device name. If this failed, there was special processing for the name /dev/null and names starting with /bin and /tmp:

```
/dev/null      NLA0:
/bin           SYS$SYSTEM:
/tmp          SYS$SCRATCH:
```

These behaviors are retained for compatibility purposes. In addition, support has been added to the C RTL for the logical name SYS\$POSIX_ROOT as an equivalent to "/".

To enable this feature for use by the C RTL, define SYS\$POSIX_ROOT as a concealed logical name. For example:

```
$ DEFINE/TRANSLATION=(CONCEALED,TERMINAL) SYS$POSIX_ROOT "$1$DKA0:[SYS0.abc.]"
```

To disable this feature:

```
$ DEFINE DECC$DISABLE_POSIX_ROOT {1, ENABLE 0, DISABLE}
```

Enabling SYS\$POSIX_ROOT results in the following behavior:

- If the existing translation of a UNIX path starting with "/" fails and SYS\$POSIX_ROOT is defined, the name is interpreted as if it starts with /sys\$posix_root.
- When converting from an OpenVMS to a UNIX style file name, and the OpenVMS name starts with "SYS\$POSIX_ROOT:", then the "SYS\$POSIX_ROOT:" is removed. For example, SYS\$POSIX_ROOT:[dirname] becomes /dirname. If the resulting name could be interpreted as a logical name or one of the special cases previously listed, the result is ./dirname instead of /dirname.

32-Bit User and Group Identifiers

The C RTL supports 32-bit User Identification (UID) and Group Identification (GID). 32-bit UID and GID support was added to some versions of the OpenVMS operating system to provide support for POSIX style identifiers (IDs). Currently, POSIX style IDs are supported only by some OpenVMS versions done for specific government agencies, but will be integrated into future OpenVMS releases.

Although OpenVMS 7.3-1 does not support POSIX style IDs, it does support 32-bit UID and GID. When an application is compiled to use 32-bit UID/GID, the UID and GID are derived from the UIC as in previous versions of the operating system. In some cases, such as with the getgroups function, more information may be returned when the application supports 32-bit GIDs.

To compile an application for 32-bit UID/GID support, define the macro __USE_LONG_GID_T. To compile an application for 16-bit UID/GID support, define the macro _DECC_SHORT_GID_T.

In addition, although you cannot compile with POSIX style IDs enabled on OpenVMS Version 7.3-1, you can run programs that were compiled with POSIX style IDs enabled on OpenVMS systems that do provide that support.

New Functions

The Compaq C RTL includes the following new or modified functions for OpenVMS Version 7.3-1:

New feature-switch-setting functions:

```
decc$feature_get_index  
decc$feature_get_name  
decc$feature_get_value  
decc$feature_set_value
```

New scanf functions:

vfprintf	vfwscanf
vscanf	vwsscanf
vsscanf	vswscanf

New functions to support 64-bit file sizes & offsets:

fseeko	ftello

Functions modified to support 64-bit file sizes & offsets:

fstat	mmap
ftruncate	stat
ftw	truncate
lseek	

New and Changed Features - Compaq C Version 6.4

The following C RTL enhancements were included in Compaq C Version 6.4 and OpenVMS Version 7.3:

- The `strptime` Function was Made XPG5-compliant.
- The limitation of eight nested directory levels was lifted (*Alpha only*).
- Improved Support for Extended File Specifications (*Alpha only*).
 - Case preservation in file names is supported (*Alpha only*).
 - Most C RTL functions now accept long OpenVMS style file names as arguments (*Alpha only*).
- Support is added for exact-case `argv` arguments (*Alpha only*).
- Files can be implicitly opened for shared access.
- There is an alternative way of translating UNIX file specifications.
- New C RTL functions are added.
- Support is added for a new signal, `SIGWINCH`.
- The socket routine `socket_fd` is changed to `decc$socket_fd`. This change is in the `<socket.h>` header file.

Programs using this function need to be amended to call `decc$socket_fd` instead of `socket_fd`. Otherwise they will get implicit function declaration.

The `strptime` function was made XPG5-compliant

The `strptime` function has been modified to be compliant with X/Open CAE Specification System Interfaces and Headers Issue 5 (commonly known as XPG5). The change for XPG5 is in how the `strptime` function processes the `%y` directive for a two-digit year within the century if no century is specified.

When a century is not otherwise specified, XPG5 requires that values for the `%y` directive in the range 69-99 refer to years in the twentieth century (1969 to 1999 inclusive), while values in the range 00-68 refer to years in the twenty-first century (2000 to 2068 inclusive). Essentially, for the `%y` directive, `strptime` became a "pivoting" function, with 69 being a pivoting year.

Before this change, the `strptime` function interpreted a two-digit year with no century as a year within twentieth century.

With OpenVMS Version 7.3, XPG5-compliant `strptime` becomes a default `strptime` function in the Compaq C RTL. However, the previous nonpivoting XPG4-compliant `strptime` function is retained for compatibility.

The pivoting is controlled by the DECC\$XPG4_STRPTIME logical name. To use the nonpivoting version of strptime, either:

- Define DECC\$XPG4_STRPTIME to any value before invoking the application.
OR
- Call the nonpivoting strptime directly as the function decc\$strptime_xpg4.

The limitation of Eight Nested Directory Levels was Lifted *(Alpha only)*

The Compaq C RTL I/O subsystem was enhanced to remove the restriction of eight nested directory levels for an ODS-5 device. This affects Compaq C RTL functions such as access, mkdir, opendir, rmdir, and stat.

Improved Support for Extended File Specifications *(Alpha only)*

The following sections describe improved Compaq C RTL support for extended file specifications.

The Compaq C RTL supports case preservation in file names *(Alpha only)*

Programs linked against the Compaq C run-time library DECC\$SHR can now preserve the case of file names on ODS level 5 disks. This applies when creating or reporting file names. By default, this feature is disabled. To enable this feature enter the following command:

```
$ DEFINE DECC$EFS_CASE_PRESERVE ENABLE
```

The following related feature logical name can also be enabled:

```
$ DEFINE DECC$EFS_CASE_SPECIAL ENABLE
```

With DECC\$EFS_CASE_SPECIAL enabled:

- File names that are in all uppercase are converted to lowercase when the name is reported in UNIX style.
- For file names that are not in all uppercase, case is preserved.

The commands to disable the preceding logical-name settings are:

```
$ DEFINE DECC$EFS_CASE_PRESERVE DISABLE  
$ DEFINE DECC$EFS_CASE_SPECIAL DISABLE
```

When enabled, DECC\$EFS_CASE_SPECIAL overrides the value of DECC\$EFS_CASE_PRESERVE.

The DECC\$EFS_CASE_PRESERVE and DECC\$EFS_CASE_SPECIAL logicals are checked only once per image activation, not on a file-by-file basis.

Most C RTL functions now accept long OpenVMS style file names as arguments *(Alpha only)*

For OpenVMS Alpha Version 7.2, some basic Compaq C RTL I/O functions (creat, stat, and the functions from the open family of functions) were enhanced to accept long OpenVMS style file names for an ODS-5 device.

For OpenVMS Alpha Version 7.3, all other Compaq C RTL functions, except chdir and the functions from the exec family of functions, were also enhanced to accept long OpenVMS style file names for an ODS-5 device.

All C RTL functions that accept or report full file specifications will process file specifications up to 4095 bytes long, subject to the rules defined for the media format. For file specifications in OpenVMS format, there are no special restrictions. In situations where a full file specification cannot be reported

because the buffer is too short, the function attempts to report the abbreviated name.

UNIX file names have the following restrictions:

- Names containing special characters, such as multiple periods, caret, or multinational characters, may be rejected.
- A function call may report failure if the output buffer is not large enough to receive the full name. For OpenVMS style names, the reported name would contain a FID-abbreviated name. There is no representation of FID-abbreviated names defined for UNIX.

The Compaq C RTL Supports Exact-Case `argv` Arguments *(Alpha only)*

Nonquoted command-line arguments passed to C and C++ programs (`argv` arguments) can now optionally have their case preserved, rather than being lowercased as in previous versions.

By default, this feature is disabled.

To enable this case preservation feature, define the logical name `DECC$ARGV_PARSE_STYLE` to "ENABLE" and set the process-level DCL parse style flag to "EXTENDED" in the process running the program:

```
$ DEFINE DECC$ARGV_PARSE_STYLE ENABLE
$ SET PROCESS/PARSE_STYLE=EXTENDED
```

Enabling this feature also ensures that the image name returned in `argv[0]` is also case-preserved.

To disable this feature, use any one of the following commands:

```
$ SET PROCESS/PARSE_STYLE=TRADITIONAL

or

$ DEFINE/SYSTEM DECC$ARGV_PARSE_STYLE DISABLE

or

$ DEASSIGN DECC$ARGV_PARSE_STYLE
```

The value of the `DECC$ARGV_PARSE_STYLE` logical is case-insensitive.

The Compaq C RTL Can Implicitly Open Files for Shared Access

The Compaq C RTL was enhanced to open all files for shared access as if the "`shr=del,get,put,upd`" option was specified in the `open*` or `creat` call.

To enable this feature, define the logical name `DECC$FILE_SHARING` to "ENABLE". The value is case-insensitive.

`DECC$FILE_SHARING` is checked only once per image activation, not on a file-by-file basis.

Alternative Way of Translating UNIX File Specs

The Compaq C RTL was enhanced to allow interpreting the leading part of a UNIX style file spec as either a subdirectory name or a device name.

As with previous releases, the default translation of `foo/bar` (UNIX style name) is `FOO:BAR` (OpenVMS style device name).

To request translation of foo/bar (UNIX style name) to [.FOO]BAR (OpenVMS style subdirectory name), define the logical name DECC\$DISABLE_TO_VMS_LOGNAME_TRANSLATION to any value. DECC\$DISABLE_TO_VMS_LOGNAME_TRANSLATION is checked only once per image activation, not on a file-by-file basis.

New Functions

The Compaq C RTL has added the following functions in OpenVMS Version 7.3:

fchown
link
utime
utimes
writev

Introduction

The ISO/ANSI C standard defines a library of functions, as well as related types and macros, to be provided with any implementation of ANSI C. The *Compaq C Language Reference Manual* describes the ANSI-conformant library features common to all *Compaq C* platforms. The *Compaq C Run-Time Library Reference Manual for OpenVMS Systems* provides a more detailed description of these routines and their use in the *OpenVMS* environment. It also documents additional header files, functions, types, and macros that are available on the *OpenVMS* system.

All library functions are declared in a *header file*. To make the contents of a header file available to your program, include the header file with an `#include` preprocessor directive. For example:

```
#include <stdlib.h>
```

Each header file contains function prototypes for a set of related functions, and defines any types and macros needed for their use.

To list the header files on *OpenVMS Alpha* systems, use the following commands:

```
$ LIBRARY/LIST ALPHA$LIBRARY:SYS$STARLET_C.TLB
$ LIBRARY/LIST ALPHA$LIBRARY:DECC$RTLDEF.TLB
$ DIR SYS$COMMON:[DECC$LIB.REFERENCE.DECC$RTLDEF]*.H;
$ DIR ALPHA$LIBRARY:*.H;
```

The first command lists the text module form of the header files for the *OpenVMS* system interfaces. The second lists the text module form of the header files for the *Compaq C* language interface. The third lists *.H header files for the *Compaq C* language interfaces. The fourth lists *.H header files for layered products and other applications.

Note

The SYS\$COMMON:[DECC\$LIB.REFERENCE.DECC\$RTLDEF] directory is only a reference area for your viewing. The compiler still looks in the *.TLB files for `#include` file searches.

To list the header files on *OpenVMS VAX* systems, use one of the following commands:

```
$ DIR 'F$TRNLN("DECC$LIBRARY_INCLUDE")'*.H;
$ DIR DECC$LIBRARY_INCLUDE:*.H;
```

On *OpenVMS* VAX systems, the following command might also find additional or duplicate header files:

```
$ DIR SYS$LIBRARY:*.*H;
```

However, duplicate files (such as `<stdio.h>`) found in `SYS$LIBRARY` probably support the VAX C Version 3.2 environment and should not be used with *Compaq C*.

Function definitions themselves are not included in the header files, but are contained in the *Compaq C* Run-Time Library (RTL) shipped with the *OpenVMS* operating system. Before using the *Compaq C* RTL, you must be familiar with the following topics:

- The linking process
- The macro substitution process
- The difference between function definitions and function calls
- The format of valid file specifications
- The *OpenVMS*-specific methods of input and output (I/O)
- The *Compaq C* for *OpenVMS* extensions and nonstandard features

A knowledge of all these topics is necessary to effectively use the *Compaq C* RTL. This chapter shows the connections between these topics and the *Compaq C* RTL. Read this chapter before any of the other chapters in this manual.

The primary purpose of the *Compaq C* RTL is to provide a means for C programs to perform I/O operations; the C language itself has no facilities for reading and writing information. In addition to I/O support, the *Compaq C* RTL also provides a means to perform many other tasks.

Chapters 2 through 11 describe the various tasks supported by the *Compaq C* RTL. The Reference Section alphabetically lists and describes all the functions and macros available to perform these tasks.

1.1 Using the *Compaq C* Run-Time Library

When working with the *Compaq C* RTL, you must be aware of some implementation specifics.

First, if you plan to use *Compaq C* RTL functions in your C programs, make sure that a function named `main` or a function that uses the `main_program` option exists in your program. For more information, see the *Compaq C Language Reference Manual* or the *Compaq C User's Guide for OpenVMS Systems*.

Second, the *Compaq C* RTL functions are executed at run time, but references to these functions are resolved at link time. When you link your program, the *OpenVMS* Linker resolves all references to *Compaq C* RTL functions by searching any shareable code libraries or object code libraries specified on the LINK command line.

You can use the *Compaq C* RTL as a shareable image or you can use the *Compaq C* RTL object libraries.

When you use the *Compaq C* RTL as a shareable image, the code for the RTL resides in an image file in `SYS$SHARE` and is shared by all *Compaq C* programs. After execution, control returns to your program. This process has a number of advantages:

- You reduce the size of a program's executable image.
- The program's image takes up less disk space.
- The program swaps in and out of memory faster due to decreased size.
- With *Compaq C* and *Compaq C++*, you no longer need to define an options file when linking your program against the shareable image. Linking against the RTL shareable image is now much simpler than it was with VAX C. In fact, it is the default method of linking to the *Compaq C* RTL.

When linking to the *Compaq C* RTL, you do not need to define any LNK\$LIBRARY logicals. In fact, you should deassign LNK\$LIBRARY because linking with the shareable image is more convenient than linking with the *Compaq C* RTL object libraries.

See your *OpenVMS*, *Compaq C*, or *Compaq C++* release notes for any supplemental information about linking with the *Compaq C* RTL.

1.2 RTL Linking Options on Alpha Systems (Alpha only)

The following sections describe several ways of linking *Compaq C* and *Compaq C++* programs with the *Compaq C* RTL on *OpenVMS Alpha* systems.

1.2.1 Linking with the Shareable Image

Most linking needs should be satisfied by using the *Compaq C* RTL shareable image DECC\$SHR.EXE in the ALPHA\$LIBRARY directory.

The shareable images VAXCRTL.EXE and VAXCRTLG.EXE do not exist on *OpenVMS Alpha* systems. The only C RTL shareable image is ALPHA\$LIBRARY:DECC\$SHR.EXE, which the linker automatically finds through IMAGELIB.OLB.

The fact that VAXCRTL*.EXE does not exist on *Alpha* systems has the following ramifications:

- You must change any existing VAX C link procedures to eliminate any references to the VAXCRTL*.EXE images. An explicit reference to DECC\$SHR.EXE is unnecessary because IMAGELIB.OLB is searched automatically by the linker (see the *OpenVMS Linker Utility Manual*).
- Because DECC\$SHR.EXE exports only prefixed universal symbols (ones that begin with DECC\$), to successfully link against it make sure you cause prefixing to occur for all *Compaq C* RTL entry points that you use.

If you use only the *Compaq C* RTL functions defined in the ANSI C Standard, all entry points will be prefixed.

If you use *Compaq C* RTL functions not defined in the ANSI C Standard, you must compile in one of two ways to ensure prefixing:

- Compile with the /PREFIX_LIBRARY_ENTRIES=ALL_ENTRIES qualifier.
- Compile with the /STANDARD=VAXC or /STANDARD=COMMON qualifier; you get /PREFIX_LIBRARY_ENTRIES=ALL_ENTRIES as the default.

To link against the shareable image, use the LINK command. For example:

```
$ LINK PROG1
```

The linker automatically searches IMAGELIB.OLB to find DECC\$SHR.EXE, and resolves all C RTL references.

1.2.2 Linking with the Object Libraries

The *Compaq C* RTL object libraries are used solely for linking programs compiled without /PREFIX=ALL.

On *OpenVMS Alpha* systems, the *Compaq C* RTL provides the following object libraries in the ALPHA\$LIBRARY directory:

- VAXCCURSE.OLB
- VAXCRTLD.OLB
- VAXCRTLT.OLB
- VAXCRTL.OLB
- VAXCRTLX.OLB
- VAXCRTLDX.OLB
- VAXCRTLTX.OLB

The object library VAXCCURSE.OLB, which provides access to the Curses functions, contains unprefix entry points that vector to the appropriate prefixed entry points.

The object libraries VAXCRTL.OLB, VAXCRTLD.OLB, VAXCRTLT.OLB, VAXCRTLX.OLB, VAXCRTLDX.OLB, and VAXCRTLTX.OLB also contain unprefix entry points that vector to the appropriate prefixed entry points, depending on the floating-point type specified by the object library used:

- VAXCRTL.OLB contains all *Compaq C* RTL routine name entry points as well as VAX G-floating double-precision, floating-point entry points.
- VAXCRTLD.OLB contains a limited support of VAX D-floating double-precision, floating-point entry points.
- VAXCRTLT.OLB contains IEEE T-floating double-precision, floating-point entry points.
- VAXCRTLX.OLB contains G-floating support and support for the /L_DOUBLE_SIZE=128 compiler qualifier.
- VAXCRTLDX.OLB contains D-floating support and support for the /L_DOUBLE_SIZE=128 compiler qualifier.
- VAXCRTLTX.OLB contains IEEE T-floating support and support for the /L_DOUBLE_SIZE=128 compiler qualifier.

/L_DOUBLE_SIZE=128 is the default.

On the LINK command, specify only one of the VAXCRTL*.OLB libraries and, if needed, the VAXCCURSE.OLB library.

In the default mode of the compiler (/STANDARD=RELAXED_ANSI89) and also in strict ANSI C mode, all calls to ANSI C standard library routines are automatically prefixed with DECC\$. With the /[NO]PREFIX_LIBRARY_ENTRIES qualifier, you can change this to prefix all *Compaq C* RTL names with DECC\$, or to not prefix any *Compaq C* RTL names. Other options are also available for this qualifier. See the /[NO]PREFIX_LIBRARY_ENTRIES qualifier in this chapter for more information.

When linking with /NOSYSSHR, if calls to the *Compaq C* RTL routines are prefixed with DECC\$, then the modules in STARLET.OLB are the only ones you need to link against. Since STARLET.OLB is automatically searched by the linker (unless the link qualifier /NOSYSLIB is used), all prefixed RTL external names are automatically resolved.

If any calls to the *Compaq C* RTL routines are not prefixed, then you need to explicitly link against VAXCRT.LIB, VAXCRTD.LIB, VAXCRTLT.LIB (or VAXCRTLX.LIB, VAXCRTLDX.LIB, or VAXCRTLDX.LIB), or VAXCCURSE.LIB, depending on which floating-point types you need, or if you want Curses functions. If you are linking with /NOSYSSHR, prefixed *Compaq C* RTL entry points are resolved in STARLET.OLB. If you are linking with /SYSSHR (the default), prefixed *Compaq C* RTL entry points are resolved in DECC\$SHR.EXE.

1.2.3 Examples

The following examples show several different ways you might want to link with the *Compaq C* RTL. See Figure 1–1 for a graphical summary of these examples.

1. Most of the time, you just want to link against the shareable image:

```
$ CC/PREFIX LIBRARY_ENTRIES=ALL_ENTRIES PROG1
$ LINK PROG1
```

The linker automatically searches IMAGELIB.OLB to find DECC\$SHR.EXE.

2. If you want to use just object libraries (to write privileged code or for ease of distribution, for example), use the /NOSYSSHR qualifier of the LINK command:

```
$ CC/PREFIX LIBRARY_ENTRIES=ALL_ENTRIES PROG1
$ LINK/NOSYSSHR PROG1
```

Prefixed RTL symbol references in the user program are resolved in the *Compaq C* RTL object library contained in STARLET.OLB.

Notes

- When linking *Compaq C* programs against the *Compaq C* RTL object libraries using the /NOSYSSHR qualifier, applications that previously linked without undefined globals may result in undefined globals for the CMA\$TIS symbols. To resolve these undefined globals, add the following line to your link options file:
SYS\$LIBRARY:STARLET.OLB/LIBRARY/INCLUDE=CMA\$TIS
 - If a program linked with the /NOSYSSHR qualifier makes a call to a routine that resides in a dynamically activated image, and the routine returns a value indicating an unsuccessful status, errno is set to ENOSYS, and vaxc\$errno is set to C\$_NOSYSSHR. The error message corresponding to C\$_NOSYSSHR is "Linking /NOSYSSHR disables dynamic image activation." An example of this situation is a program linked with /NOSYSSHR that makes a call to a socket routine.
-

- When compiling with prefixing disabled, in order to use object libraries that provide alternate implementations of C RTL functions, you need to use the VAXC*.OLB object libraries. In this case, compile and link as follows:

```
$ CC/NOPREFIX LIBRARY ENTRIES PROG1
$ LINK PROG1, MYLIB/LIBRARY, ALPHA$LIBRARY:VAXCRTLX.OLB/LIBRARY
```

Unprefixed *Compaq C* RTL symbol references in the user program are resolved in MYLIB and in VAXCRTL.OLB.

Prefixed *Compaq C* RTL symbol references in VAXCRTLX.OLB are resolved in DECC\$SHR.EXE through IMAGELIB.OLB.

In this same example, to get IEEE T-floating double-precision floating-point support, you might use the following compile and link commands:

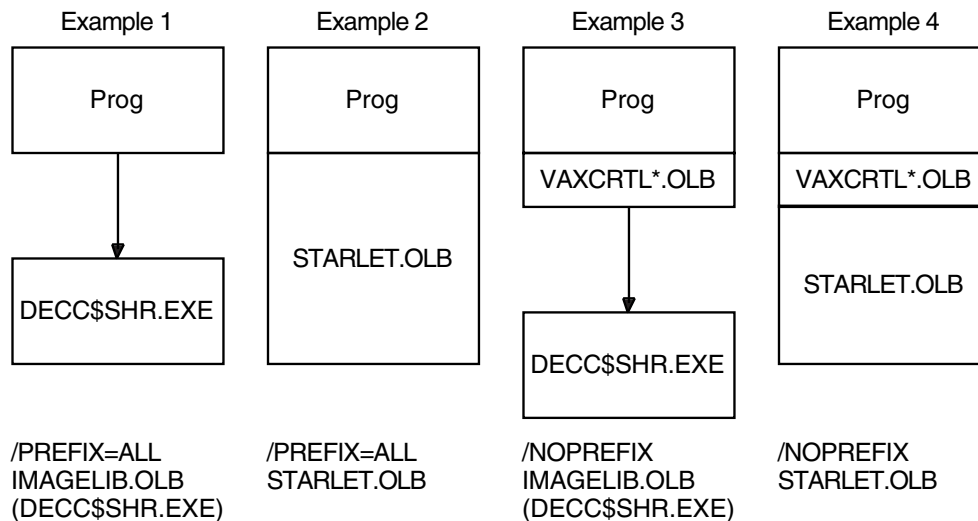
```
$ CC/NOPREFIX LIBRARY ENTRIES/FLOAT=IEEE FLOAT PROG1
$ LINK PROG1, MYLIB/LIBRARY, ALPHA$LIBRARY:VAXCRTLX.OLB/LIBRARY
```

- Combining examples 2 and 3, you might want to use just the object libraries (for writing privileged code or for ease of distribution) and use an object library that provides C RTL functions. In this case, compile and link as follows:

```
$ CC/NOPREFIX LIBRARY ENTRIES PROG1
$ LINK/NOSYSSHR PROG1, MYLIB/LIBRARY, ALPHA$LIBRARY:VAXCRTLX.OLB/LIBRARY
```

Prefixed *Compaq C* RTL symbol references in VAXCRTL.OLB are resolved in STARLET.OLB.

Figure 1–1 Linking with the *Compaq C* RTL on *OpenVMS Alpha* Systems



ZK-6045A-GE

1.3 RTL Linking Options on VAX Systems (VAX only)

Both the VAX C RTL and the *Compaq C* RTL can coexist on your *OpenVMS VAX* system. The VAX C RTL supports existing VAX C applications. The *Compaq C* RTL supports ANSI-compliant *Compaq C* and *Compaq C++*, as well as other components of the *OpenVMS* environment. The *Compaq C* RTL also provides a mechanism for thread safety and performance improvements.

Applications developed with VAX C will continue to use the VAX C RTL. However, you can relink VAX C applications to use the *Compaq C* RTL instead. This lets you take advantage of the new features of the *Compaq C* RTL and solve potential interoperability problems in complex applications that incorporate both the VAX C RTL and the *Compaq C* RTL. Existing applications that are relinked to use the *Compaq C* RTL should be carefully tested for possible problems resulting from the differences in behavior between the VAX C RTL and the *Compaq C* RTL. See the applicable *Compaq C* release notes and *OpenVMS* release notes for more information.

The following sections describe several ways of linking *Compaq C* programs with the *Compaq C* RTL and VAX C RTL on *OpenVMS* VAX systems.

1.3.1 Linking with the *Compaq C* RTL

The *Compaq C* RTL provides a new set of files with different names from the VAX C RTL files. If you want to link with the *Compaq C* RTL, you need to change your link procedures to use the new file names. The following sections describe linking with the *Compaq C* RTL files.

1.3.1.1 Linking with the *Compaq C* RTL Shareable Images

Most linking needs should be satisfied by using the *Compaq C* RTL shareable image DECC\$SHR.EXE in the SYS\$LIBRARY directory. Use this linking method for programs that are written entirely in *Compaq C* or *Compaq C++* code; that is, with no VAX C object modules.

Because DECC\$SHR.EXE exports only prefixed universal symbols (ones that begin with DECC\$), to successfully link against it make sure you cause prefixing to occur for all *Compaq C* RTL entry points.

If you use only the *Compaq C* RTL functions defined in the ANSI C Standard, all entry points will be prefixed.

If you use *Compaq C* RTL functions not defined in the ANSI C Standard, you must compile in one of two ways to ensure prefixing:

- Compile with the /PREFIX_LIBRARY_ENTRIES=ALL_ENTRIES qualifier.
- Compile with the /STANDARD=VAXC or /STANDARD=COMMON qualifier; you get /PREFIX_LIBRARY_ENTRIES=ALL_ENTRIES as the default.

Then link against the shareable image using the LINK command. For example:

```
$ LINK PROG1
```

If you are using the VAX C compiler and you want to link with DECC\$SHR.EXE, you must link to one of the following files:

```
VAXC2DECC.EXE
VAXCG2DECC.EXE
```

You link with them as follows:

```
$ LINK PROG1,TT:/OPTIONS
SYS$LIBRARY:VAXC2DECC/SHARE
Ctrl/Z
```

Use the G-floating version, VAXCG2DECC.EXE, if you compiled with the /G_FLOAT or /FLOAT=G_FLOAT qualifier.

1.3.1.2 Linking with or Providing Your Own Shareable Images

Most linking needs for an application using a shareable image are handled by a straightforward link command, regardless of whether the shared image uses *Compaq C*, *VAX C* or some other programming language.

For example, assume that SHARE1.EXE is a shareable image linked with VAXCRTL.EXE. Also assume that your program, PROG1, is compiled with *Compaq C* and, therefore, references prefixed names for C RTL functions. You can then use the following commands:

```
$ LINK PROG1, SYS$INPUT:/OPTIONS
MYDISK:[TEXT]SHARE1.EXE/SHARE
```

If PROG1 does not use prefixed names, the link could result in link conflicts. If this occurs, see Section 1.3.2.

1.3.1.3 Linking with the *Compaq C* RTL Object Libraries

The *Compaq C* RTL object libraries are used primarily for linking with the /NOSYSSHR qualifier.

On *OpenVMS VAX* systems, the *Compaq C* RTL provides the following object libraries in the SYS\$LIBRARY directory:

- DECCCURSE.OLB
- DECCRTL.OLB
- DECCRTL.OLB

As with *VAX C*, if you specify more than one object library on the LINK command, you must do so in the order listed.

You use these object libraries in the same way that you would use the *VAX C* RTL object libraries VAXCRTL.OLB, VAXCRTL.OLB, and VAXCCURSE.OLB. For example:

```
$ ! Link a D-float program
$ LINK PROG1, SYS$LIBRARY:DECCRTL.OLB/LIBRARY
$ !
$ ! Link a G-float program
$ LINK PROG2, SYS$LIBRARY:DECCRTL.OLB/LIBRARY, -
_$ SYS$LIBRARY:DECCRTL.OLB/LIBRARY
_$ !
$ ! Link a D-float, Curses program
$ LINK PROG1, SYS$LIBRARY:DECCURSE.OLB/LIBRARY, -
_$ SYS$LIBRARY:DECCRTL.OLB/LIBRARY
```

Note

When linking to the *Compaq C* RTL object libraries, you do not need to define any LNK\$LIBRARY logicals. In fact, you must deassign LNK\$LIBRARY when linking with the .OLB libraries; otherwise, you might see "multiply defined symbols" errors.

In general, you should deassign LNK\$LIBRARY because pointing this logical to the *Compaq C* RTL object libraries interferes with *VAX C* development.

1.3.1.4 Linking with the *Compaq C* RTL Object Libraries /NOSYSSHR

If you want to link your program with the *Compaq C* RTL object libraries using the /NOSYSSHR qualifier, you must specify /INCLUDE=CMA\$TIS for the object library. For OpenVMS VAX Version 7.3 and higher, you must specify /INCLUDE=(CMA\$TIS,CMA\$TIS_VEC). Otherwise, several symbols will be undefined and the resulting image will not execute.

In order to add this qualifier, you cannot use the LNK\$LIBRARY logicals to link with the *Compaq C* RTL. You must use a linker options file or list the *Compaq C* RTL object library on the command line. For example:

```
$ LINK/NOSYSSHR PROG1, SYS$LIBRARY:DECCRTL.OLB/LIBRARY/INCLUDE=CMA$TIS
```

```
$ LINK/NOSYSSHR PROG1, -  
_ $ SYS$LIBRARY:DECCRTL.OLB/LIBRARY/INCLUDE=(CMA$TIS,CMA$TIS_VEC) (OpenVMS V7.3  
and higher)
```

Notes

- When linking *Compaq C* programs against the *Compaq C* RTL object libraries using the /NOSYSSHR qualifier, applications that previously linked without undefined globals may result in undefined globals for the CMA\$TIS symbols. To resolve these undefined globals, add the following line to your link options file:

```
SYS$LIBRARY:STARLET.OLB/LIBRARY/INCLUDE=CMA$TIS
```

```
SYS$LIBRARY:STARLET.OLB/LIBRARY/INCLUDE=(CMA$TIS,CMA$TIS_VEC)  
(OpenVMS V7.3  
and higher)
```

- If a program linked with the /NOSYSSHR qualifier makes a call to a routine that resides in a dynamically activated image, and the routine returns a value indicating an unsuccessful status, errno is set to ENOSYS, and vaxc\$errno is set to C\$_NOSYSSHR. The error message corresponding to C\$_NOSYSSHR is "Linking /NOSYSSHR disables dynamic image activation." An example of this situation is a program linked with /NOSYSSHR that makes a call to a socket routine.

1.3.2 Resolving Link-Time Conflicts with Multiple C RTLs

This section describes the use of interoperability tools to resolve link-time conflicts when using multiple C RTLs.

When migrating to the *Compaq C* RTL, multiple C RTLs will likely be needed to link an application. One C RTL might be explicitly linked against. A second C RTL might not be explicitly linked against, but brought into the link by means of a shareable image. For example, when developing a Motif program using *Compaq C*, the application must be linked against the *Compaq C* RTL and against the Motif images. Motif currently brings the VAX C RTL into the link.

Problems encountered when linking with multiple C RTLs are a result of the *OpenVMS* linker resolving symbol references in the image being linked by searching the transitive closure of shareable images and libraries. That is, when linking with a shareable image, the linker searches that shareable image and all shareable images referenced in that shareable image. So when linking with

VAXCRTL.EXE and with an image linked with VAXCRTL.G.EXE, the linker will find two instances of all the C RTL symbols (one in VAXCRTL and one in VAXCRTL.G), and report a conflict.

The object libraries do not conflict with routine names, but do conflict with the global symbols. Because VAX C implements global symbols as global overlaid psects, the linker attempts to connect all the instances of a C-generated psect with the same name. For example, a reference to stdin in the user program is connected with the psect of the same name in VAXCRTL.OLB. However, a shareable image that was linked with VAXCRTL.OLB also has a psect of the same name; this results in an error because the linker cannot connect those two definitions of the psect stdin.

Three interoperability tools are provided with the *Compaq C* compiler and in a separate *Compaq C/C++* RTL Run-Time Components kit to resolve link-time conflicts:

- VAXC\$LCL.OPT
- VAXC\$EMPTY.EXE
- DECC\$EMPTY.EXE

These tools work by hiding the conflicting symbols from one of the C RTLs being linked. Which tool is required depends on what C RTLs are used by the main application and the shareable image.

Table 1–1 shows typical C RTL conflicts and the interoperability tool required to resolve it. In the table, VAXCRTL.EXE refers to either VAXCRTL.EXE or VAXCRTL.G.EXE.

Table 1–1 Linking Conflicts

Linker Message	Type of Conflict	Tool Needed
LINK-E-MULSHRPSC	VAXCRTL.OLB/VAXCRTL.EXE	VAXC\$LCL.OPT
LINK-E-SHRPSC LNG	VAXCRTL.OLB/DECCRTL.OLB	VAXC\$LCL.OPT
LINK-E-MULSHRPSC, LINK-E-SHRPSC LNG	DECCRTL.OLB/VAXCRTL.EXE	VAXC\$LCL.OPT
None.	DECCRTL.OLB/DECC\$SHR.EXE	DECC\$EMPTY
LINK-W-MULDEF	VAXCRTL.EXE/VAXCRTL.G.EXE	VAXC\$EMPTY
LINK-W-MULDEF	VAXC2DECC.EXE/VAXCRTL.EXE	VAXC\$EMPTY

1.3.2.1 Using VAXC\$LCL.OPT

VAXC\$LCL.OPT is required when building any shareable image linked with the VAX C RTL object library or *Compaq C* RTL object library.

If the shareable image is built without using VAXC\$LCL.OPT, the C RTL global symbols are visible in the shareable image and cause linker conflicts when users of the image link against it. For example:

```
%LINK-E-MULSHRPSC, psect C$$TRNS VALUES defined in
      shareable image IMAGE1.EXE; is multiply defined in
      shareable image SYS$LIBRARY:VAXCRTL.EXE;1
-LINK-E-NOIMGFIL, image file not created
```


In this example, the shareable image IMAGE1 uses VAXCRTL.OLB, and the image being linked uses VAXCRTL.EXE. For a successful link, relink the shareable image using VAXC\$LCL.OPT:

```
$ LINK/SHARE IMAGE1.OBJ, IMAGE1.OPT/OPTIONS, SYS$LIBRARY:VAXCRTL/LIBRARY, -
_$ SYS$LIBRARY:VAXC$LCL.OPT/OPTIONS
```

The following message also indicates a conflict involving the VAX C RTL object library:

```
%LINK-E-SHRPSCNG, Psect STDIN has length of 8
      in module C$EXTERNDATA file SYS$LIBRARY:DECCRTL.OLB;2
      which exceeds length of 4 in shareable image IMAGE1.EXE;
-LINK-E-NOIMGFIL, image file not created
```

In this example, the shareable image IMAGE1 uses VAXCRTL.OLB, and the image being linked uses DECCRTL.OLB. For a successful link, relink the shareable image using VAXC\$LCL.OPT.

If the shareable image cannot be relinked (as in the case of a third-party shareable image), then the interoperability tool can be applied to the main image. If the main image is being linked against DECCRTL.OLB, then apply VAXC\$LCL.OPT to the link of the main image.

If the main image is being linked against VAXCRTL.EXE, the only solution is to get the shareable image fixed, because applying any of the interoperability tools to the link of the main image will result in an unsuccessful link.

1.3.2.2 Using VAXC\$EMPTY.EXE

Use VAXC\$EMPTY.EXE to link a main application with both VAXC2DECC.EXE (or VAXCG2DECC.EXE) and a shareable image linked with VAXCRTL.EXE (or VAXCRTLG.EXE). Using VAXC\$EMPTY.EXE hides all the global symbols in the VAXCRTL*.EXE shareable image to prevent conflicts with VAXC2DECC.EXE or VAXCG2DECC.EXE.

Also use VAXC\$EMPTY.EXE to link an application with both VAXCRTL.EXE and a shareable image linked with VAXCRTLG.EXE (or vice versa).

When there is a conflict between C RTL shareable images, the linker produces large numbers of messages similar to the following:

```
%LINK-W-MULDEF, symbol ACOS multiply defined
      in module VAXCRTL file SYS$COMMON:[SYSLIB]VAXCRTL.EXE;18
```

In this example, the shareable image is linked with VAXCRTL.EXE, and the main program is linked with VAXC2DECC.EXE.

The solution is to define the VAXCRTL logical to point to VAXC\$EMPTY.EXE before linking the main program:

```
$ DEFINE/USER VAXCRTL SYS$LIBRARY:VAXC$EMPTY.EXE
$ LINK/EXEC=MAIN_IMAGE MAIN_PROG,OBJ1,OBJ2,...,SYS$INPUT:/OPTIONS
IMAGE1/SHARE
VAXCRTL/SHARE
[Ctrl/Z]
```

Note the following about this solution:

- Your linker options file cannot reference SYS\$LIBRARY:VAXCRTL; it must reference only VAXCRTL.
- Do not link explicitly against VAXC\$EMPTY.EXE, or your application will neither link nor run correctly.

- Do not leave the VAXCRTL pointing to VAXC\$EMPTY.EXE, or your application will not run correctly.
- The DEFINE/USER command is used to ensure that the logical definition is removed after execution of the LINK command. Make sure that no commands intervene between the DEFINE/USER command and the LINK command.

Follow the same process when linking against VAXCRTL.G.EXE by defining the VAXCRTL.G logical to point to VAXC\$EMPTY.EXE.

1.3.2.3 Using DECC\$EMPTY.EXE

The DECC\$EMPTY.EXE interoperability tool allows a program to use the *Compaq C* object library even when the program links with a shareable image that was linked with DECC\$SHR.EXE.

If DECC\$EMPTY.EXE is not used during the link, all *Compaq C* RTL references from the main program will be resolved in DECC\$SHR.EXE, not in the object library. There is no linker message that indicates this fact.

For example, if IMAGE1 is linked against DECC\$SHR, and the following link is performed, then the main image will not contain any *Compaq C* RTL object modules. All C RTL references from the main program are resolved in DECC\$SHR:

```
$ LINK/EXEC=MAIN_IMAGE MAIN_PROG,OBJ1,...,SYS$INPUT:/OPTIONS
IMAGE1/SHARE
SYS$LIBRARY:DECCRTL/LIBRARY
Ctrl/Z
```

By defining the DECC\$SHR logical to point to DECC\$EMPTY.EXE immediately before the link, all references to C RTL symbols from the main program are resolved in the *Compaq C* RTL object library. For example:

```
$ DEFINE/USER DECC$SHR SYS$LIBRARY:DECC$EMPTY.EXE
$ LINK/EXEC=MAIN_IMAGE MAIN_PROG,OBJ1,...,SYS$INPUT:/OPTIONS
IMAGE1/SHARE
SYS$LIBRARY:DECCRTL/LIBRARY
```

Note the following about this solution:

- Do not link explicitly against DECC\$EMPTY.EXE, or your application will neither link correctly nor run correctly.
- Do not leave the DECC\$SHR logical pointing to DECC\$EMPTY.EXE, or your application will not run correctly.
- The DEFINE/USER command is used to ensure that the logical definition is removed after execution of the LINK command. Make sure that no commands intervene between the DEFINE/USER command and the LINK command.

1.3.3 Linking Examples for *Compaq C* or *Compaq C++* Code Only

The following examples show the different ways you might want to link *Compaq C* only or *Compaq C++* only programs with the *Compaq C* RTL on *OpenVMS VAX* systems:

1. Most of the time, you just want to link against the shareable image:

```
$ CC/DECC/PREFIX_LIBRARY_ENTRIES=ALL_ENTRIES PROG1
$ LINK PROG1
```

The linker automatically searches IMAGELIB.OLB to find DECC\$SHR.EXE.

2. If you want to use just object libraries (to write privileged code or for ease of distribution, for example), use the /NOSYSSHR qualifier of the LINK command:

```
$ CC/DECC/PREFIX LIBRARY ENTRIES=ALL ENTRIES PROG1
$ LINK/NOSYSSHR PROG1, SYS$LIBRARY:DECCRTL.OLB/LIBRARY/INCL=CMA$TIS

$ LINK/NOSYSSHR PROG1, -
_$ SYS$LIBRARY:DECCRTL.OLB/LIBRARY/INCL=(CMA$TIS,CMA$TIS_VEC) (OpenVMS V7.3
and higher)
```

Prefixed *Compaq C* RTL symbol references in the user program are resolved in STARLET.OLB.

3. When compiling with prefixing disabled, in order to use object libraries that provide alternate implementations of C RTL functions, you need to use the DECC*.OLB object libraries. In this case, compile and link as follows:

```
$ CC/DECC/NOPREFIX LIBRARY ENTRIES PROG1
$ LINK PROG1, MYLIB/LIBRARY, -
_$ SYS$LIBRARY:DECCRTL.OLB/LIBRARY
```

Unprefixed *Compaq C* RTL symbol references in the user program are resolved in MYLIB and DECCRTL.OLB. The unprefixed names reference prefixed names resolved in DECC\$SHR.EXE.

You can link with any valid combination of DECCRTL.OLB, DECCRTL.G.OLB, and DECCCURSE.OLB. In this same example, to get G-floating double-precision, floating-point support, you might use the following compile and LINK commands:

```
$ CC/DECC/NOPREFIX LIBRARY ENTRIES/FLOAT=G FLOAT PROG1
$ LINK PROG1, MYLIB/LIBRARY, SYS$LIBRARY:DECCRTL.G.OLB/LIBRARY, -
_$ SYS$LIBRARY:DECCRTL.OLB/LIBRARY
```

4. Combining examples 2 and 3, you might want to use just the object libraries (for writing privileged code or for ease of distribution) and use an object library that provides C RTL functions. In this case, compile and link as follows:

```
$ CC/DECC/NOPREFIX LIBRARY ENTRIES PROG1
$ LINK/NOSYSSHR PROG1, MYLIB/LIBRARY, -
_$ SYS$LIBRARY:DECCRTL.OLB/LIBRARY
```

1.3.4 Linking Examples for VAX C and *Compaq C* Code Combined

You might have programs that combine VAX C and *Compaq C* (or *Compaq C++*) code. The following examples show different ways to link such programs with the *Compaq C* RTL on *OpenVMS* VAX systems. These examples correspond to the examples in Section 1.3.3.

1. To link against the shareable image, specify the VAXC2DECC.EXE shareable image:

```
$ CC/DECC PROG1
$ CC/VAXC PROG2
$ LINK PROG1, PROG2, TT:/OPTIONS
SYS$LIBRARY:VAXC2DECC.EXE/SHARE
[Ctrl/Z]
```

Prefixed C RTL calls from PROG1 are resolved in DECC\$SHR. Unprefixed C RTL calls from PROG2 are resolved in VAXC2DECC.EXE, which transfers them to DECC\$SHR.

2. If you want to use just object libraries (to write privileged code or for ease of distribution, for example), use the /NOSYSSHR qualifier of the LINK command:

```
$ CC/DECC PROG1
$ CC/VAXC PROG2
$ LINK/NOSYSSHR PROG1, PROG2, SYS$LIBRARY:DECCRTL.OLB/LIBRARY/INCL=CMA$TIS
```

All C RTL calls from both PROG1 and PROG2 are resolved in DECCRTL.OLB.

3. When compiling with prefixing disabled, in order to use object libraries that provide alternate implementations of C RTL functions, you need to use the DECC*.OLB object libraries. In this case, compile and link as follows:

```
$ CC/DECC/NOPREFIX_LIBRARY_ENTRIES PROG1
$ CC/VAXC PROG2
$ LINK PROG1, PROG2, MYLIB/LIBRARY, -
_$SYS$LIBRARY:DECCRTL.OLB/LIBRARY/INCL=CMA$TIS
```

Unprefixed *Compaq C* RTL symbol references in the user program are resolved in MYLIB and DECCRTL.OLB.

4. Combining examples 2 and 3, you might want to use just the object libraries (for writing privileged code or for ease of distribution) and use an object library that provides C RTL functions. In this case, compile and link as follows:

```
$ CC/DECC/NOPREFIX_LIBRARY_ENTRIES PROG1
$ CC/VAXC PROG2
$ LINK/NOSYSSHR PROG1, PROG2, MYLIB/LIBRARY, -
_$ SYS$LIBRARY:DECCRTL.OLB/LIBRARY /INCL=CMA$TIS
```

1.3.5 Linking with the VAX C RTL /NOSYSSHR

This section applies to programs running on *OpenVMS VAX* Version 6.0 or higher.

For programs that currently link with the VAX C RTL object libraries using the /NOSYSSHR qualifier, you must specify /INCLUDE=CMA\$TIS for the object library. Otherwise, several symbols will be undefined and the resulting image will not execute. In order to add this qualifier, you cannot use the LNK\$LIBRARY logicals to link with the VAX C RTL object libraries. You must use a linker options file or list the VAX C RTL object libraries on the command line. For example:

```
$ LINK/NOSYSSHR PROG1, SYS$LIBRARY:VAXCRTL.OLB/LIBRARY/INCLUDE=CMA$TIS
```

1.4 Compaq C RTL Function Prototypes and Syntax

After learning how to link object modules and include header files, you must learn how to reference *Compaq C* functions in your program. The remaining chapters in this manual provide detailed descriptions of the *Compaq C* RTL functions.

1.4.1 Function Prototypes

In all chapters, the syntax describing each function follows the standard convention for defining a function. This syntax is called a *function prototype* (or just *prototype*). The prototype is a compact representation of the order of a function's arguments (if any), the types of the arguments, and the type of the value returned by a function. The use of prototypes is recommended.

If the return value of the function cannot be easily represented by a C data-type keyword, look for a description of the return values in the explanatory text. The prototype descriptions provide insight into the functionality of the function. These descriptions may not describe how to call the function in your source code.

For example, consider the prototype for the `feof` function:

```
#include <stdio.h>
int feof(FILE *file_ptr);
```

This syntax shows the following information:

- The `feof` prototype resides in the `<stdio.h>` header file. To use `feof`, you must include this header file. (Declaring *Compaq C* RTL functions yourself is not recommended.)
- The `feof` function returns a value of data type `int`.
- There is one argument, *file_ptr*, that is of type "pointer to FILE". `FILE` is defined in the `<stdio.h>` header file.

To use `feof` in a program, include `<stdio.h>` anywhere before the function call to `feof`, as in the following example:

```
#include <stdio.h>                /* Include Standard I/O    */
main()
{
    FILE *infile;                 /* Define a file pointer */
    .
    .
    .
    while ( ! feof(infile) )      /* Call the function feof */
    {                             /* Until EOF reached      */
        .                         /* Perform file operations */
        .
        .
    }
}
```

1.4.2 Syntax Conventions for Function Prototypes

Since some library functions take a varying number of parameters, syntax descriptions for function prototypes adhere to the following conventions:

- Ellipses (`...`) are used to indicate a varying number of parameters.
- In cases where the type of a parameter may vary, its type is not shown in the syntax.

Consider the `printf` syntax description:

```
#include <stdio.h>
int printf(const char *format_specification, ... );
```

The syntax description for `printf` shows that you can specify one or more optional parameters. The remaining information about `printf` parameters is in the description of the function.

1.4.3 UNIX Style File Specifications

The *Compaq C* RTL functions and macros often manipulate files. One of the major portability problems is the different file specifications used on various systems. Since many C applications are ported to and from UNIX systems, it is convenient for all compilers to be able to read and understand UNIX system file specifications.

The following file specification conversion functions are included in the *Compaq C* RTL to assist in porting C programs from UNIX systems to *OpenVMS* systems:

- decc\$match_wild
- decc\$translate_vms
- decc\$fix_time
- decc\$to_vms
- decc\$from_vms

Note

These DECC\$ routine names replace the SHELL\$ routine names previously available with the VAX C RTL.

The advantage of including these file specification conversion functions in the *Compaq C* RTL is that you do not have to rewrite C programs containing UNIX system file specifications. *Compaq C* can translate most valid UNIX system file specifications to *OpenVMS* file specifications.

Please note the differences between the UNIX system and *OpenVMS* file specifications, as well as the method used by the RTL to access files. For example, the RTL accepts a valid *OpenVMS* specification and most valid UNIX file specifications, but the RTL cannot accept a combination of both. Table 1–2 shows the differences between UNIX system and *OpenVMS* system file specification delimiters.

Table 1–2 UNIX and *OpenVMS* File Specification Delimiters

Description	<i>OpenVMS</i> System	UNIX System
Node delimiter	::	!/
Device delimiter	:	/
Directory path delimiter	[]	/
Subdirectory delimiter	[.]	/
File extension delimiter	.	.
File version delimiter	;	Not applicable

For example, Table 1–3 shows the formats of two valid specifications and one invalid specification.

Table 1–3 Valid and Invalid UNIX and *OpenVMS* File Specifications

System	File Specification	Valid/Invalid
<i>OpenVMS</i>	BEATLE::DBA0:[MCCARTNEY]SONGS.LIS	Valid
UNIX	beatle!/usr1/mccartney/songs.lis	Valid
—	BEATLE::DBA0:[MCCARTNEY.C]/songs.lis	Invalid

When *Compaq C* translates file specifications, it looks for both *OpenVMS* and UNIX system file specifications. Consequently, there may be differences between how *Compaq C* translates UNIX system file specifications and how UNIX systems translate the same UNIX file specification.

For example, if the two methods of file specification are combined, as in Table 1–3, *Compaq C* RTL can interpret [MCCARTNEY.C]/songs.lis as either [MCCARTNEY]songs.lis or [C]songs.lis. Therefore, when *Compaq C* encounters a mixed file specification, an error occurs.

UNIX systems use the same delimiter for the device name, the directory names, and the file name. Due to the ambiguity of UNIX file specifications, *Compaq C* may not translate a valid UNIX system file specification according to your expectations.

For instance, the *OpenVMS* system equivalent of /bin/today can be either [BIN]TODAY or [BIN.TODAY]. *Compaq C* can make the correct interpretation only from the files present. If a file specification conforms to UNIX system file name syntax for a single file or directory, it is converted to the equivalent *OpenVMS* file name if one of the following conditions is true:

- If the specification corresponds to an existing *OpenVMS* directory, it is converted to that directory name. For example, /dev/dir/sub is converted to DEV:[DIR.SUB] if DEV:[DIR.SUB] exists.
- If the specification corresponds to an existing *OpenVMS* file name, it is converted to that file name. For example, /dev/dir/file is converted to DEV:[DIR]FILE if DEV:[DIR]FILE exists.
- If the specification corresponds to a nonexistent *OpenVMS* file name, but the given device and directory exist, it is converted to a file name. For example, /dev/dir/file is converted to DEV:[DIR]FILE if DEV:[DIR] exists.

Note

Beginning with *OpenVMS* Version 7.3, you can instruct the *Compaq C* RTL to interpret the leading part of a UNIX style file spec as either a subdirectory name or a device name.

As with previous releases, the default translation of foo/bar (UNIX style name) is FOO:BAR (*OpenVMS* style device name).

To request translation of foo/bar (UNIX style name) to [.FOO]BAR (*OpenVMS* style subdirectory name), define the logical name DECC\$DISABLE_TO_VMS_LOGNAME_TRANSLATION to any value. DECC\$DISABLE_TO_VMS_LOGNAME_TRANSLATION is checked only once per image activation, not on a file-by-file basis. Defining this logical affects not only the decc\$to_vms function, but all *Compaq C* RTL

functions that accept both UNIX style and OpenVMS style file names as an argument.

In the UNIX system environment, you reference files with a numeric file descriptor. Some file descriptors reference Standard I/O devices; some descriptors reference actual files. If the file descriptor belongs to an unopened file, the *Compaq C* RTL opens the file. *Compaq C* equates file descriptors with the following *OpenVMS* logical names:

File Descriptor	<i>OpenVMS</i> Logical	Meaning
0	SYS\$INPUT	Standard input
1	SYS\$OUTPUT	Standard output
2	SYS\$ERROR	Standard error

1.4.4 Extended File Specifications

The ODS-5 volume structure provides enhanced support for mixed UNIX and OpenVMS style file names. It supports long file names, allows the use of a wider range of characters within file names, and preserves case within file names. With OpenVMS Alpha Version 7.3-1, the C RTL has greatly improved support of ODS-5 characters, with 250 of the 256 characters supported, as opposed to only 214 supported previously. Also, file names without file types can now be accessed.

To enable the new support, you must define one or more C RTL feature logical names. These names include the following:

```
DECC$EFS_CHARSET
DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION
DECC$FILENAME_UNIX_NO_VERSION
DECC$FILENAME_UNIX_REPORT
DECC$REaddir_DROPDOTNOTYPE
DECC$RENAME_NO_INHERIT
```

See Section 1.6 for more information on these and other feature logical names.

1.5 Feature-Test Macros for Header-File Control

Feature-test macros provide a means for writing portable programs. They ensure that the *Compaq C* RTL symbolic names used by a program do not clash with the symbolic names supplied by the implementation.

The *Compaq C* RTL header files are coded to support the use of a number of feature-test macros. When an application defines a feature-test macro, the *Compaq C* RTL header files supply the symbols and prototypes defined by that feature-test macro and nothing else. If a program does not define such a macro, the *Compaq C* RTL header files define symbols without restriction.

The feature-test macros supported by the *Compaq C* RTL fall into three broad categories for controlling the visibility of symbols in header files according to the following:

- Standards
- Multiple-version support
- Compatibility

1.5.1 Standards Macros

The *Compaq C* RTL implements parts of the following standards:

- X/Open CAE Specification, System Interfaces and Headers, Issue 4, Version 2, also known as XPG4 V2.
- X/Open CAE Specification, System Interfaces and Headers, Issue 4, also known as XPG4.
- Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API)—Amendment 2: Threads Extension [C Language], also known as POSIX 1003.1c-1995 or IEEE 1003.1c-1995.
- ISO/IEC 9945-2:1993 - Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities, also known as ISO POSIX-2.
- ISO/IEC 9945-1:1990 - Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Programming Interface (API) (C Language), also known as ISO POSIX-1.
- ANSI/ISO/IEC 9899:1999 - The C99 standard, published by ISO in December, 1999 and adopted as an ANSI standard in April, 2000.
- ISO/IEC 9899:1990-1994 - Programming Languages - C, Amendment 1: Integrity, also known as ISO C, Amendment 1.
- ISO/IEC 9899:1990 - Programming Languages - C, also known as ISO C. The normative part is the same as X3.159-1989, American National Standard for Information Systems - Programming Language C, also known as ANSI C.

1.5.2 Selecting a Standard

You can define a feature-test macro to select each standard. You can do this either with a `#define` preprocessor directive in your C source before the inclusion of any header file, or with the `/DEFINE` qualifier on the CC command line.

Table 1–4 lists and describes the *Compaq C* RTL feature-test macros that control standards support.

Table 1–4 Feature Test Macros - Standards

Macro Name	Standard Selected	Other Standards Implied	Description
<code>_XOPEN_SOURCE_EXTENDED</code>	XPG4 V2	XPG4, ISO POSIX-2, ISO POSIX-1, ANSI C	Makes visible XPG4-extended features, including traditional UNIX based interfaces not previously adopted by X/Open.

(continued on next page)

Table 1–4 (Cont.) Feature Test Macros - Standards

Macro Name	Standard Selected	Other Standards Implied	Description
<code>_XOPEN_SOURCE</code>	XPG4	ISO POSIX-2, ISO POSIX-1, ANSI C	Makes visible XPG4 standard symbols and causes <code>_POSIX_C_SOURCE</code> to be set to 2 if it is not already defined with a value greater than 2. ^{1, 2}
<code>_POSIX_C_SOURCE==199506</code>	IEEE 1003.1c-1995	ISO POSIX-2, ISO POSIX-1, ANSI C	Header files defined by ANSI C make visible those symbols required by IEEE 1003.1c-1995.
<code>_POSIX_C_SOURCE == 2</code>	ISO POSIX-2	ISO POSIX-1, ANSI C	Header files defined by ANSI C make visible those symbols required by ISO POSIX-2 plus those required by ISO POSIX-1.
<code>_POSIX_C_SOURCE == 1</code>	ISO POSIX-1	ANSI C	Header files defined by ANSI C make visible those symbols required by ISO POSIX-1.
<code>__STDC_VERSION__==199409</code>	ISO C amdt 1	ANSI C	Makes ISO C amendment 1 symbols visible.
<code>_ANSI_C_SOURCE</code>	ANSI C	—	Makes ANSI C standard symbols visible.

¹Where the ISO C Amendment 1 includes symbols not specified by XPG4, defining `__STDC_VERSION__ == 199409` and `_XOPEN_SOURCE` (or `_XOPEN_SOURCE_EXTENDED`) selects both ISO C and XPG4 APIs. Conflicts that arise when compiling with both XPG4 and ISO C Amendment 1 resolve in favor of ISO C Amendment 1.

²Where XPG4 extends the ISO C Amendment 1, defining `_XOPEN_SOURCE` or `_XOPEN_SOURCE_EXTENDED` selects ISO C APIs as well as the XPG4 extensions available in the header file. This mode of compilation makes XPG4 extensions visible.

Features not defined by one of the previously named standards are considered *Compaq C* extensions and are selected by not defining any standards-related, feature-test macros.

If you do not explicitly define feature test macros to control header file definitions, you implicitly include all defined symbols as well as *Compaq C* extensions.

1.5.3 Interactions with the /STANDARD Qualifier

The /STANDARD qualifier selects the dialect of the C language supported.

With the exception of /STANDARD=ANSI89 and /STANDARD=ISOC94, the selection of C dialect and the selection of *Compaq C* RTL APIs to use are independent choices. All other values for /STANDARD cause the entire set of APIs to be available, including extensions.

Specifying /STANDARD=ANSI89 restricts the default API set to the ANSI C set. In this case, to select a broader set of APIs, you must also specify the appropriate feature-test macro. To select the ANSI C dialect and all APIs, including extensions, undefine `__HIDE_FORBIDDEN_NAMES` before including any header file.

Compiling with /STANDARD=ISOC94 sets `__STDC_VERSION__` to 199409. Conflicts that arise when compiling with both XPG4 and ISO C Amendment 1 resolve in favor of ISO C Amendment 1. XPG4 extensions to ISO C Amendment 1 are selected by defining `_XOPEN_SOURCE`.

The following examples help clarify these rules:

- The `fdopen` function is an ISO POSIX-1 extension to `<stdio.h>`. Therefore, `<stdio.h>` defines `fdopen` only if one or more of the following is true:
 - The program including it is not compiled in strict ANSI C mode (/STANDARD=ANSI89).
 - `_POSIX_C_SOURCE` is defined as 1 or greater.
 - `_XOPEN_SOURCE` is defined.
 - `_XOPEN_SOURCE_EXTENDED` is defined.
- The `popen` function is an ISO POSIX-2 extension to `<stdio.h>`. Therefore, `<stdio.h>` defines `popen` only if one or more of the following is true:
 - The program including it is not compiled in strict ANSI C mode (/STANDARD=ANSI89).
 - `_POSIX_C_SOURCE` is defined as 2 or greater.
 - `_XOPEN_SOURCE` is defined.
 - `_XOPEN_SOURCE_EXTENDED` is defined.
- The `getw` function is an X/Open extension to `<stdio.h>`. Therefore, `<stdio.h>` defines `getw` only if one or more of the following is true:
 - The program is not compiled in strict ANSI C mode (/STANDARD=ANSI89).
 - `_XOPEN_SOURCE` is defined.
 - `_XOPEN_SOURCE_EXTENDED` is defined.
- The X/Open Extended symbolic constants `_SC_PAGESIZE`, `_SC_PAGE_SIZE`, `_SC_ATEXIT_MAX`, and `_SC_IOV_MAX` were added to `<unistd.h>` to support the `sysconf` function. However, these constants are not defined by `_POSIX_C_SOURCE`.

The `<unistd.h>` header file defines these constants only if a program does not define `_POSIX_C_SOURCE` and does define `_XOPEN_SOURCE_EXTENDED`.

If `_POSIX_C_SOURCE` is defined, these constants are not visible in `<unistd.h>`. Note that `_POSIX_C_SOURCE` is defined only for programs compiled in strict ANSI C mode.
- The `fgetname` function is a *Compaq C* RTL extension to `<stdio.h>`. Therefore, `<stdio.h>` defines `fgetname` only if the program is not compiled in strict ANSI C mode (/STANDARD=ANSI89).
- The macro `_PTHREAD_KEYS_MAX` is defined by POSIX 1003.1c-1995. This macro is made visible in `<limits.h>` when compiling for this standard with `_POSIX_C_SOURCE == 199506` defined, or by default when compiling without any standards-defining, feature-test macros.
- The macro `WCHAR_MAX` defined in `<wchar.h>` is required by ISO C Amendment 1 but not by XPG4. Therefore:
 - Compiling for ISO C Amendment 1 makes this symbol visible, but compiling for XPG4 compliance does not.
 - Compiling for both ISO C Amendment 1 and XPG4 makes this symbol visible.

Similarly, the functions `wcsftime` and `wcstok` in `<wchar.h>` are defined slightly differently by the ISO C Amendment 1 and XPG4:

- Compiling for ISO C Amendment 1 makes the ISO C Amendment 1 prototypes visible.
- Compiling for XPG4 compliance makes the XPG4 prototypes visible.
- Compiling for both ISO C Amendment 1 and XPG4 selects the ISO C prototypes because conflicts resulting from this mode of compilation resolve in favor of ISO C.
- Compiling without any standard selecting feature test macros makes ISO C Amendment 1 features visible.

So in this example, compiling with no standard-selecting feature-test macros makes `WCHAR_MAX` and the ISO C Amendment 1 prototypes for `wcsftime` and `wcstok` visible.

- The `wcswidth` and `wcwidth` functions are XPG4 extensions to ISO C Amendment 1. Their prototypes are in `<wchar.h>`.

These symbols are visible if:

- Compiling for XPG4 compliance by defining `_XOPEN_SOURCE` or `_XOPEN_SOURCE_EXTENDED`.
- Compiling for DEC C Version 4.0 compatibility or on pre-*OpenVMS* Version 7.0 systems.
- Compiling with no standard selecting feature test macros.
- Compiling for both ISO C Amendment 1 and XPG4 compliance because these symbols are XPG4 extensions to ISO C Amendment 1.

Compiling for strict ISO C Amendment 1 does not make them visible.

1.5.4 Multiple-Version-Support Macro

By default, the header files enable APIs in the *Compaq C* RTL provided by the version of the operating system on which the compilation occurs. This is accomplished by the predefined setting of the `__VMS_VER` macro, as described in the *Compaq C User's Guide for OpenVMS Systems*. For example, compiling on *OpenVMS* Version 6.2 causes only *Compaq C* RTL APIs from Version 6.2 and earlier to be made available.

Another example of the use of the `__VMS_VER` macro is support for the 64-bit versions of *Compaq C* RTL functions available with *OpenVMS Alpha* Version 7.0 and higher. In all header files, functions that provide 64-bit support are conditionalized so that they are visible only if `__VMS_VER` indicates a version of *OpenVMS* that is greater than or equal to 7.0.

To target an older version of the operating system, do the following:

1. Define a logical `DECC$SHR` to point to the old version of `DECC$SHR`. The compiler uses a table from `DECC$SHR` to perform routine name prefixing.
2. Define `__VMS_VER` appropriately, either with the `/DEFINE` qualifier or with a combination of the `#undef` and `#define` preprocessor directives. With `/DEFINE`, you may need to disable the warning regarding redefinition of a predefined macro.

Targeting a newer version of the operating system might not always be possible. For some versions, you can expect that the new DECC\$SHR.EXE will require new features of the operating system that are not present. For such versions, the defining if the logical DECC\$SHR in Step 1 would cause the compilation to fail.

To override the value of `__VMS_VER`, define `__VMS_VER_OVERRIDE` on the compiler command line. Defining `__VMS_VER_OVERRIDE` without a value sets `__VMS_VER` to the maximum value.

1.5.5 Compatibility Modes

The following predefined macros are used to select header-file compatibility with previous versions of DEC C) or the *OpenVMS* operating system:

- `_DECC_V4_SOURCE`
- `_VMS_V6_SOURCE`

There are two types of incompatibilities that can be controlled in the header files:

- To conform to standards, some changes are source-code incompatible but binary compatible. To select DEC C Version 4.0 source compatibility, use the `_DECC_V4_SOURCE` macro.
- Other changes to conform to standards introduce a binary or run-time incompatibility.

In general, programs that recompile get new behaviors. In these cases, use the `_VMS_V6_SOURCE` feature test macro to retain previous behaviors.

However, for the `exit`, `kill`, and `wait` functions, the *OpenVMS* Version 7.0 changes to make these routines ISO POSIX-1 compliant were considered too incompatible to become the default. Therefore, in these cases the default behavior is the same as on pre-*OpenVMS* Version 7.0 systems. To access the versions of these routines that comply with ISO POSIX-1, use the `_POSIX_EXIT` feature test macro.

The following examples help clarify the use of these macros:

- To conform to the ISO POSIX-1 standard, typedefs for the following have been added to `<types.h>`:

```
dev_t      off_t
gid_t      pid_t
ino_t      size_t
mode_t     ssize_t
nlink_t    uid_t
```

Previous development environments using a version of DEC C earlier than Version 5.2 may have compensated for the lack of these typedefs in `<types.h>` by adding them to another module. If this is the case on your system, then compiling with the `<types.h>` provided with DEC C Version 5.2 might cause compilation errors.

To maintain your current environment and include the DEC C Version 5.2 `<types.h>`, then compile with `_DECC_V4_SOURCE` defined. This will omit incompatible references from the DEC C Version 5.2 headers. In `<types.h>`, for example, the previously listed typedefs will not be visible.

- As of *OpenVMS* Version 7.0, the *Compaq C* RTL `getuid` and `geteuid` functions are defined to return an *OpenVMS* UIC (user identification code) that contains both the group and member portions of the UIC. In previous

versions of the DEC C RTL, these functions returned only the member number from the UIC code.

Note that the prototypes for `getuid` and `geteuid` in `<unistd.h>` (as required by the ISO POSIX-1 standard) and in `<unixlib.h>` (for *Compaq C* RTL compatibility) have not changed. By default, newly compiled programs that call `getuid` and `geteuid` get the new definitions. That is, these functions will return an *OpenVMS* UIC.

To let programs retain the pre-*OpenVMS* Version 7.0 behavior of `getuid` and `geteuid`, compile with the `_VMS_V6_SOURCE` feature-test macro defined.

- As of *OpenVMS* Version 7.0, the *Compaq C* RTL `exit` function is defined with ISO POSIX-1 semantics. As a result, the input status argument to `exit` takes a number between 0 and 255. (Prior to this, `exit` could take an *OpenVMS* condition code in its status parameter.)

By default, the behavior for `exit` on *OpenVMS* systems is the same as before — `exit` accepts an *OpenVMS* condition code. To enable the ISO POSIX-1 compatible `exit` function, compile with the `_POSIX_EXIT` feature-test macro defined.

1.5.6 Curses and Socket Compatibility Macros

The following feature-test macros are used to control the Curses and Socket subsets of the *Compaq C* RTL library:

- `_BSD44_CURSES`
This macro selects the Curses package from the 4.4BSD Berkeley Software Distribution.
- `_VMS_CURSES`
This macro selects a Curses package based on the VAX C compiler. This is the default Curses package.
- `_SOCKADDR_LEN`
This macro is used to select 4.4BSD-compatible and XPG4 V2-compatible socket interfaces. These interfaces require support in your underlying TCP/IP software. Contact your TCP/IP vendor to inquire if the version of TCP/IP software you run supports 4.4BSD sockets.

Strict XPG4 V2 compliance requires the 4.4BSD-compatible socket interface. Therefore, if `_XOPEN_SOURCE_EXTENDED` is defined on *OpenVMS* Version 7.0 or higher, `_SOCKADDR_LEN` is defined to be 1.

The following examples help clarify the use of these macros:

- Symbolic constants like `AE`, `AL`, `AS`, `AM`, `BC`, which represent pointers to termcap fields used by the BSD Curses package, are only visible in `<curses.h>` if `_BSD44_CURSES` is defined.
- The `<socket.h>` header file defines a 4.4BSD `sockaddr` structure only if `_SOCKADDR_LEN` or `_XOPEN_SOURCE_EXTENDED` is defined. Otherwise, `<socket.h>` defines a pre-4.4BSD `sockaddr` structure. If `_SOCKADDR_LEN` is defined and `_XOPEN_SOURCE_EXTENDED` is not defined, the `<socket.h>` header file also defines an `osockaddr` structure, which is a 4.3BSD `sockaddr` structure to be used for compatibility purposes. Since XPG4 V2 does not define an `osockaddr` structure, it is not visible in `_XOPEN_SOURCE_EXTENDED` mode.

1.5.7 2-Gigabyte File Size Macro

The C RTL provides support for compiling applications to use file sizes and offsets that are two gigabytes and larger. This is accomplished by allowing file offsets of 64-bit integers.

The `fseeko` and `ftello` functions, which have the same behavior as `fseek` and `ftell`, accept or return values of type `off_t`, which allows for a 64-bit variant of `off_t` to be used.

C RTL functions `lseek`, `mmap`, `ftuncate`, `truncate`, `stat`, `fstat`, and `ftw` can also accommodate a 64-bit file offset.

The new 64-bit interfaces can be selected at compile time by defining the `_LARGEFILE` feature macro.

1.5.8 32-bit UID and GID Macros

The C RTL supports 32-bit User Identification (UID) and Group Identification (GID). When an application is compiled to use 32-bit UID/GID, the UID and GID are derived from the UIC as in previous versions of the operating system. Use the following macros to control UID/GID size:

- `__USE_LONG_GID_T`
To compile an application for 32-bit UID/GID support, define the `__USE_LONG_GID_T` macro.
- `_DECC_SHORT_GID_T`
To compile an application for 16-bit UID/GID support, define the `_DECC_SHORT_GID_T` macro.

1.6 Enabling Compaq C RTL Features Using Feature Logical Names

The C RTL provides an extensive list of feature switches that can be enabled or disabled using `DECC$` logical names. These switches affect the behavior of a C application at run time.

The feature switches introduce new behaviors and also preserve old behaviors that have been deprecated.

You enable most features by setting a logical name to `ENABLE` and disable a feature by setting the logical name to `DISABLE`:

```
$ DEFINE DECC$feature ENABLE
$ DEFINE DECC$feature DISABLE
```

Some features logical names can be set to a numeric value. For example:

```
$ DEFINE DECC$PIPE_BUFFER_SIZE 65536
```

Note

Older features from earlier releases of the C Run-Time Library were documented as supplying *any equivalence string* to enable the features. While this is correct, Compaq recommends that you use "ENABLE" for all features enabled using logical names. This avoids possible side effects if the feature switch is enhanced to allow both enabling and disabling with the logical name defined.

In particular, avoid the following values for any feature previously documented as accepting *any equivalence string*:

- DISABLE
- 0 (zero)
- F
- N

Table 1–5 lists the C RTL feature logical names, grouped by the type of features they control.

Table 1–5 C RTL Feature Logical Names

Feature Logical Name	Default
Performance Optimizations	
DECC\$ENABLE_GETENV_CACHE	DISABLE
DECC\$LOCALE_CACHE_SIZE	0
DECC\$TZ_CACHE_SIZE	2
Legacy Behaviors	
DECC\$V62_RECORD_GENERATION	DISABLE
DECC\$XPG4_STRPTIME	DISABLE
DECC\$THREAD_DATA_AST_SAFE	DISABLE
File Attributes	
DECC\$DEFAULT_LRL	32767
DECC\$DEFAULT_UDF_RECORD	DISABLE
DECC\$FIXED_LENGTH_SEEK_TO_EOF	DISABLE
Mailboxes	
DECC\$MAILBOX_CTX_STM	DISABLE
Changes for UNIX Conformance	
DECC\$STRTOL_ERANGE	DISABLE
DECC\$VALIDATE_SIGNAL_IN_KILL	DISABLE
DECC\$SELECT_IGNORES_INVALID_FD	DISABLE

(continued on next page)

Table 1–5 (Cont.) C RTL Feature Logical Names

Feature Logical Name	Default
General UNIX Enhancements	
DECC\$ARGV_PARSE_STYLE	DISABLE
DECC\$PIPE_BUFFER_SIZE	512
DECC\$STDIO_CTX_EOL	DISABLE
DECC\$USE_RAB64	DISABLE
Enhancements for UNIX Style File Names	
DECC\$DISABLE_TO_VMS_LOGNAME_TRANSLATION	DISABLE
DECC\$EFS_CHARSET	DISABLE
DECC\$FILENAME_UNIX_NO_VERSION	DISABLE
DECC\$FILENAME_UNIX_REPORT	DISABLE
DECC\$READDIR_DROPDOTNOTYPE	DISABLE
DECC\$RENAME_NO_INHERIT	DISABLE
Enhancements for UNIX Style File Attributes	
DECC\$EFS_FILE_TIMESTAMPS	DISABLE
DECC\$EXEC_FILEATTR_INHERITANCE	DISABLE
DECC\$FILE_OWNER_UNIX	DISABLE
DECC\$FILE_PERMISSION_UNIX	DISABLE
DECC\$FILE_SHARING	DISABLE
UNIX Compliance Mode	
DECC\$FILENAME_UNIX_ONLY	DISABLE
DECC\$DETACHED_CHILD_PROCESS	DISABLE
New Behaviors for POSIX Conformance	
DECC\$POSIX_SEEK_STREAM_FILE	DISABLE
DECC\$UMASK	RMS default
File Name Handling	
DECC\$READDIR_KEEPPDOTDIR	DISABLE
DECC\$EFS_CASE_PRESERVE	DISABLE
DECC\$EFS_CASE_SPECIAL	DISABLE
DECC\$UNIX_PATH_BEFORE_LOGNAME	DISABLE
DECC\$DISABLE_POSIX_ROOT	DISABLE

An alphabetic listing and description of the C RTL feature logical names follows. Unless otherwise stated, the feature logicals are enabled with **ENABLE** and disabled with **DISABLE**.

DECC\$ARGV_PARSE_STYLE

With DECC\$ARGV_PARSE_STYLE enabled, case is preserved in command line arguments when the process has been set up for extended DCL parsing using SET PROCESS/PARSE_STYLE=EXTENDED.

DECC\$ARGV_PARSE_STYLE must be defined externally as a logical name or set in a function called using the LIB\$INITIALIZE mechanism because it is evaluated before function main is called.

DECC\$DEFAULT_LRL

DECC\$DEFAULT_LRL specifies the default value for the RMS attribute for longest record length. The default value 32767 is the largest record size supported by RMS.

Default: 32767

Maximum: 32767

DECC\$DEFAULT_UDF_RECORD

With DECC\$DEFAULT_UDF_RECORD enabled, file access mode defaults to RECORD instead of STREAM mode for all files except STREAMLF.

DECC\$DETACHED_CHILD_PROCESS

With DECC\$DETACHED_CHILD_PROCESS enabled, child processes created using vfork and exec are created as detached processes instead of subprocesses.

This feature has only limited support. In some cases the console cannot be shared between the parent process and the detached process, which can cause exec to fail.

DECC\$DISABLE_POSIX_ROOT

With DECC\$DISABLE_POSIX_ROOT enabled, support for the POSIX root directory defined by SYS\$POSIX_ROOT is disabled.

With DECC\$DISABLE_POSIX_ROOT disabled, the SYS\$POSIX_ROOT logical name is interpreted as the equivalent of the file path "/". If a UNIX path starting with "/" is given and the value after the leading slash cannot be translated as a logical name, SYS\$POSIX_ROOT is used as the parent directory for the specified UNIX file path.

More Detail:

The C RTL supports a UNIX style root that behaves like a real directory. This allows such actions as:

```
% cd /
% mkdir /dirname
% tar -xvf tarfile.tar /dirname
% ls /
```

Previously, the C RTL did not recognize "/" as a directory name. The normal processing for a file path starting with "/" was to interpret the first element as a logical name or device name. If this failed, there was special processing for the name /dev/null and names starting with /bin and /tmp:

/dev/null	NLA0:
/bin	SYS\$SYSTEM:
/tmp	SYS\$SCRATCH:

These behaviors are retained for compatibility purposes. In addition, support has been added to the C RTL for the logical name SYS\$POSIX_ROOT as an equivalent to "/".

To enable this feature for use by the C RTL, define SYS\$POSIX_ROOT as a concealed logical name. For example:

```
$ DEFINE/TRANSLATION=(CONCEALED,TERMINAL) SYS$POSIX_ROOT "$1$DKA0:[SYS0.abc.]"
```

To disable this feature:

```
$ DEFINE DECC$DISABLE_POSIX_ROOT {1, ENABLE 0, DISABLE}
```

Enabling SYS\$POSIX_ROOT results in the following behavior:

- If the existing translation of a UNIX path starting with "/" fails and SYS\$POSIX_ROOT is defined, the name is interpreted as if it starts with /sys\$posix_root.
- When converting from an OpenVMS to a UNIX style file name, and the OpenVMS name starts with "SYS\$POSIX_ROOT:", then the "SYS\$POSIX_ROOT:" is removed. For example, SYS\$POSIX_ROOT:[dirname] becomes /dirname. If the resulting name could be interpreted as a logical name or one of the special cases previously listed, the result is ./dirname instead of /dirname.

DECC\$DISABLE_TO_VMS_LOGNAME_TRANSLATION

With DECC\$DISABLE_TO_VMS_LOGNAME_TRANSLATION enabled, the conversion routine decc\$to_vms will only treat the first element of a UNIX style name as a logical name if there is a leading slash "/".

DECC\$EFS_CASE_PRESERVE

With DECC\$EFS_CASE_PRESERVE enabled, case is preserved for file names on ODS level 5 disks.

With DECC\$EFS_CASE_PRESERVE disabled, UNIX style file names are always reported in lowercase.

However, note that enabling DECC\$EFS_CASE_SPECIAL overrides the setting for DECC\$EFS_CASE_PRESERVE.

DECC\$EFS_CASE_SPECIAL

With DECC\$EFS_CASE_SPECIAL enabled, case is preserved only for file names containing lowercase. If an element of a file name contains all uppercase letters, it is reported in all lowercase in UNIX style.

When enabled, DECC\$EFS_CASE_SPECIAL overrides the value of DECC\$EFS_CASE_PRESERVE.

DECC\$EFS_CHARSET

With DECC\$EFS_CHARSET enabled, UNIX names can contain ODS-5 extended characters. Support includes multiple dots and all ASCII characters in the range 0 to 255, except the following:

```
<NUL>  
/  
"  
*  
?
```

Unless `DECC$FILENAME_UNIX_ONLY` is enabled, some characters can be interpreted as OpenVMS characters depending on context. They are:

```
:
[
<
^
;
```

`DECC$EFS_CHARSET` might be necessary for existing applications that make assumptions about file names based on the presence of certain characters, because the following non-standard and undocumented C RTL extensions do not work when EFS extended character-set support is enabled:

- `$HOME` is interpreted as the user's login directory
With `DECC$EFS_CHARSET` enabled, `$HOME` is treated literally and may be in an OpenVMS or UNIX style file name.
- `~name` is interpreted as the login directory for user name
With `DECC$EFS_CHARSET` enabled, `~name` is treated literally and can be in an OpenVMS or UNIX style file name.
- Wild card regular expressions in the form `[a-z]`
With `DECC$EFS_CHARSET` enabled, square brackets are acceptable in OpenVMS and UNIX style file names. For instance, in a function such as `open, abc[a-z]ef.txt` is interpreted as a UNIX style name equivalent to the OpenVMS style name `abc^[a-z^]ef.txt`, and `[a-z]bc` is interpreted as an OpenVMS style name equivalent to the UNIX style name `/sys$disk/a-z/bc`.

With `DECC$EFS_CHARSET` enabled, the following encoding for EFS extended characters is supported when converting from an OpenVMS style file name to a UNIX style file name:

- All ODS-2 compatible names
- All encoding for 8-bit characters, either as single byte or using two-digit hexadecimal form `^ab`. In a UNIX path these are always represented as a single byte.
- Encoding for DEL (`^7F`)
- The following characters when preceded by a caret:
space ! , _ & ' () + @ { } ; # [] % ^ = \$ - ~ .
- The following characters when not preceded by a caret:
\$ - ~ .
- The implementation supports the conversion from OpenVMS to UNIX needed for functions `readdir`, `ftw`, `getname`, `fgetname`, `getcwd`, and others.

DECC\$EFS_FILE_TIMESTAMPS

With `DECC$EFS_FILE_TIMESTAMPS` enabled, `stat` and `fstat` report new ODS-5 access time (`st_atime`), attribute revision time (`st_ctime`) and modification time (`st_mtime`) for files on ODS-5 volumes that have the extended file times enabled using `SET VOLUME/VOLUME=ACCESS_DATES`.

If `DECC$EFS_FILE_TIMESTAMPS` is disabled, or the volume is not ODS-5, or the volume does not have support for these additional times enabled, `st_ctime` continues to be the file creation time and `st_atime` the same as the `st_mtime`.

Functions `utime` and `utimes` support these ODS-5 times in the same way as `stat`.

DECC\$ENABLE_GETENV_CACHE

The C RTL supplements the list of environment variables in the `environ` table with all logical names and DCL symbols available to the process.

By default, whenever `getenv` is called for a name not in the `environ` table, an attempt is made to resolve this as a logical name and, if this fails, as a DCL symbol.

With `DECC$ENABLE_GETENV_CACHE` enabled, once a logical name or DCL name has been successfully translated, its value is stored in a cache. When the same name is requested in a future call to `getenv`, the value is returned from the cache instead of reevaluating the logical name or DCL symbol.

DECC\$EXEC_FILEATTR_INHERITANCE

With `DECC$EXEC_FILEATTR_INHERITANCE` enabled, the current file pointer and the file open mode is passed to the child process in `exec` calls if the child is a C program.

With this logical name disabled, the child process does not inherit append mode or the file position.

DECC\$FILENAME_UNIX_ONLY

With `DECC$FILENAME_UNIX_ONLY` enabled, file names are never interpreted as OpenVMS style names. This prevents any interpretation of the following as OpenVMS special characters:

: [^

DECC\$FILENAME_UNIX_NO_VERSION

With `DECC$FILENAME_UNIX_NO_VERSION` enabled, OpenVMS version numbers are not supported in UNIX style file names.

With `DECC$FILENAME_UNIX_NO_VERSION` disabled, in UNIX style names, version numbers are reported preceded by a dot (.).

DECC\$FILENAME_UNIX_REPORT

With `DECC$FILENAME_UNIX_REPORT` enabled, all file names are reported in UNIX style unless the caller specifically selects OpenVMS style. This applies to `getpwnam`, `getpwuid`, `argv[0]`, `getname`, and `fgetname`.

With `DECC$FILENAME_UNIX_REPORT` disabled, unless specified in the function call, file names are reported in OpenVMS style.

DECC\$FILE_OWNER_UNIX

With `DECC$FILE_OWNER_UNIX` enabled, the owner for a new file or directory is always based on the effective UIC. When an earlier version of the file exists, the owner for the new file is inherited from the earlier version.

With `DECC$FILE_OWNER_UNIX` disabled, the owner for a new file is set following OpenVMS rules and may inherit the owner from the parent directory.

DECC\$FILE_PERMISSION_UNIX

With `DECC$FILE_PERMISSION_UNIX` enabled, the file permissions for new files and directories are set according to the file creation mode and `umask`. This includes mode `0777`. When an earlier version of the file exists, the file permissions for the new file are inherited from the earlier version. This mode sets `DELETE` permission for a new directory when `WRITE` permission is enabled.

With `DECC$FILE_PERMISSION_UNIX` disabled, modes 0 and 0777 indicate using RMS default protection or protection from the previous version of the file. Permissions for new directories also follow OpenVMS rules, including disabling DELETE permissions.

DECC\$FILE_SHARING

With `DECC$FILE_SHARING` enabled, all files are opened with full sharing enabled (`FAB$M_DEL` | `FAB$M_GET` | `FAB$M_PUT` | `FAB$M_UPD`). This is set as a logical OR with any sharing mode specified by the caller.

DECC\$FIXED_LENGTH_SEEK_TO_EOF

With `DECC$FIXED_LENGTH_SEEK_TO_EOF` enabled, `lseek`, `fseeko`, and `fseek` with the *direction* parameter set to `SEEK_END` will position relative to the last byte in the file for files with fixed-length records.

With `DECC$FIXED_LENGTH_SEEK_TO_EOF` disabled, `lseek`, `fseek`, and `fseeko` when called with `SEEK_EOF` on files with fixed-length records, will position relative to the end of the last record in the file.

DECC\$LOCALE_CACHE_SIZE

`DECC$LOCALE_CACHE_SIZE` defines how much memory, in bytes, to allocate for caching locale data. The default value is 0, which disables the locale cache.

Default: 0

Maximum: 2147483647

DECC\$MAILBOX_CTX_STM

By default, an open on a local mailbox that is not a pipe treats mailbox records as having a record attribute of `FAB$M_CR`.

With `DECC$MAILBOX_CTX_STM` enabled, the record attribute `FAB$M_CR` is not set.

DECC\$PIPE_BUFFER_SIZE

The system default buffer size of 512 bytes for pipe write operations can limit performance and generate extra line feeds when handling messages longer than 512 bytes.

`DECC$PIPE_BUFFER_SIZE` allows a larger default buffer size to be used for pipe functions such as `pipe` and `popen`. A value of 512 to 65024 bytes can be specified.

If `DECC$PIPE_BUFFER_SIZE` is not specified, the default buffer size 512 is used.

Default: 512

Minimum: 512

Maximum: 65536

DECC\$POSIX_SEEK_STREAM_FILE

With `DECC$POSIX_SEEK_STREAM_FILE` enabled, positioning beyond end-of-file on `STREAM` files does not write to the file until the next write. If the write is beyond the current end-of-file, this positions beyond the old end-of-file, and the start position for the write is filled with zeros.

With `DECC$POSIX_SEEK_STREAM_FILE` disabled, positioning beyond end-of-file will immediately write zeros to the file from the current end-of-file to the new position.

DECC\$READDIR_DROPDOTNOTYPE

With `DECC$READDIR_DROPDOTNOTYPE` enabled, `readdir` when reporting files in UNIX style only reports the trailing dot for files with no file type when the file name contains a dot.

With this logical name disabled, all files without a file type are reported with a trailing dot.

DECC\$READDIR_KEEPPDOTDIR

The default behavior when reporting files in UNIX style from `readdir` is to report directories without a file type.

With `DECC$READDIR_KEEPPDOTDIR` enabled, directories are reported in UNIX style with a file type of `".DIR"`.

DECC\$RENAME_NO_INHERIT

With `DECC$RENAME_NO_INHERIT` enabled, the new name for the file does not inherit anything from the old name. The new name must be specified completely.

With `DECC$RENAME_NO_INHERIT` disabled, the new file name inherits missing components of the file name such as the device, directory, file type, and version from the old file, in the same way as the `DCL RENAME` command.

DECC\$SELECT_IGNORES_INVALID_FD

With `DECC$SELECT_IGNORES_INVALID_FD` enabled, `select` fails with `errno` set to `EBADF` when an invalid file descriptor is specified in one of the descriptor sets.

With `DECC$SELECT_IGNORES_INVALID_FD` disabled, `select` ignores invalid file descriptors.

DECC\$STDIO_CTX_EOL

With `DECC$STDIO_CTX_EOL` enabled, writing to `stdout` and `stderr` for stream access is deferred until a terminator is seen or the buffer is full.

With `DECC$STDIO_CTX_EOL` disabled, each `fwrite` generates a separate write, which for mailbox and record files generates a separate record.

DECC\$STRTOL_ERANGE

With `DECC$STRTOL_ERANGE` enabled, the `strtol` behavior for an `ERANGE` error is corrected to consume all remaining digits in the string.

With `DECC$STRTOL_ERANGE` disabled, the legacy behavior of leaving the pointer at the failing digit is preserved.

DECC\$THREAD_DATA_AST_SAFE

The C RTL has a mode that allocates storage for thread-specific data allocated by threads at non-AST level separate for data allocated for ASTs. In this mode, each access to thread-specific data requires a call to `LIB$AST_IN_PROG`, which can add significant overhead when accessing thread-specific data in the C RTL.

The alternate mode protects thread-specific data only if another function has it locked. This protects data that is in use within the C RTL, but does not protect the caller from an AST changing the data pointed to.

This latter mode is now the C RTL default for the `strtok`, `ecvt`, and `fcvt` functions.

You can select the legacy AST safe mode by enabling `DECC$THREAD_DATA_AST_SAFE`.

DECC\$TZ_CACHE_SIZE

`DECC$TZ_CACHE_SIZE` specifies the number of time zones that can be held in memory.

Default: 2

Maximum: 2147483647

DECC\$UMASK

`DECC$UMASK` specifies the default value for the permission mask `umask`. By default, a parent C program sets the `umask` from the RMS default permissions for the process. A child process inherits the parent's value for `umask`.

To enter the value as an octal value, add the leading zero; otherwise, it is translated as a decimal value. For example:

```
$ DEFINE DECC$UMASK 026
```

Maximum: 0777

DECC\$UNIX_PATH_BEFORE_LOGNAME

With `DECC$UNIX_PATH_BEFORE_LOGNAME` enabled, when translating a UNIX file name not starting with a leading slash (/), an attempt is made to match this to a file or directory in the current directory. If this is not found and the name is valid as a logical name in an OpenVMS file name, an attempt is made to translate the logical name and, if found, is used as part of the resulting file name.

Enabling `DECC$UNIX_PATH_BEFORE_LOGNAME` overrides the setting for `DECC$DISABLE_TO_VMS_LOGNAME_TRANSLATION`.

DECC\$USE_RAB64

With `DECC$USE_RAB64` enabled, open functions allocate a RAB64 structure instead of the traditional RAB structure.

This provides latent support for file buffers in 64-bit memory.

DECC\$VALIDATE_SIGNAL_IN_KILL

With `DECC$VALIDATE_SIGNAL_IN_KILL` enabled, a signal value that is in the range 0 to `_SIG_MAX` but is not supported by the C RTL generates an error with `errno` set to `EINVAL`, which makes the behavior the same as for `raise`.

With this logical name disabled, validation of signals is restricted to checking that the signal value is in the range 0 to `_SIG_MAX`. If `sys$sigprc` fails, `errno` is set based on `sys$sigprc` exit status.

DECC\$V62_RECORD_GENERATION

OpenVMS Versions 6.2 and higher can output record files using different rules.

With `DECC$V62_RECORD_GENERATION` enabled, the output mechanism follows the rules used for OpenVMS Version 6.2.

DECC\$XPG4_STRPTIME

XPG5 support for `strptime` introduces pivoting year support so that years in the range 0 to 68 are in the 21st century, and years in the range 69-99 are in the 20th century.

With `DECC$XPG4_STRPTIME` enabled, XPG5 support for the pivoting year is disabled and all years in the range 0 to 99 are in the current century.

1.7 32-Bit User and Group Identifiers

When supported in versions of the OpenVMS operating system, POSIX style identifiers refers to the User Identification (UID), Group Identification (GID), and Process Group. The scope includes real and effective identifiers.

The support for POSIX style identifiers in the Compaq C RTL requires 32-bit user and group ID support and also depends on features in the base version of OpenVMS. Currently, POSIX style IDs are supported only by some OpenVMS versions done for specific government agencies, but will be integrated into future OpenVMS releases.

OpenVMS Version 7.3-1 does not support POSIX style IDs, but it does support 32-bit UID/GID. When an application is compiled to use 32-bit UID/GID, the UID and GID are derived from the UIC as in previous versions of the operating system. In some cases, such as with the `getgroups` function, more information may be returned when the application supports 32-bit GIDs.

On OpenVMS Version 7.3-1, to compile an application for 32-bit UID/GID support, define the macro `__USE_LONG_GID_T`. To compile an application for 16-bit UID/GID support, define the macro `_DECC_SHORT_GID_T`.

In addition, although you cannot compile with POSIX style IDs enabled on OpenVMS Version 7.3-1, you can run programs that were compiled with POSIX style IDs enabled on OpenVMS systems that do provide that support.

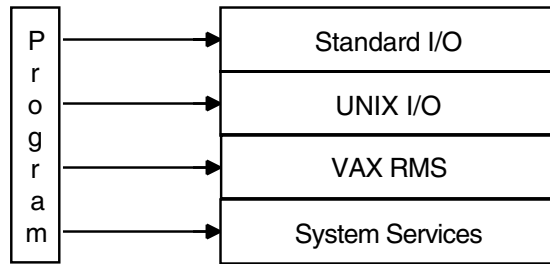
1.8 Input and Output on *OpenVMS* Systems

After you learn how to link with the *Compaq C* RTL and call *Compaq C* functions and macros, you can use the *Compaq C* RTL for its primary purpose: input/output (I/O).

Since every system has different methods of I/O, familiarize yourself with the *OpenVMS*-specific methods of file access. In this way, you will be equipped to predict functional differences when porting your source program from one operating system to another.

Figure 1-2 shows the I/O methods available with the *Compaq C* RTL. The *OpenVMS* system services communicate directly with the *OpenVMS* operating system, so they are closest to the operating system. The *OpenVMS Record Management Services* (RMS) functions use the system services, which manipulate the operating system. The *Compaq C* Standard I/O and UNIX I/O functions and macros use the RMS functions. Since the *Compaq C* RTL Standard I/O and UNIX I/O functions and macros must go through several layers of function calls before the system is manipulated, they are furthest from the operating system.

Figure 1–2 I/O Interface from C Programs



ZK–0493–GE

The C programming language was developed on the UNIX operating system, and the Standard I/O functions were designed to provide a convenient method of I/O that would be powerful enough to be efficient for most applications, and also be portable so that the functions could be used on any system running C language compilers.

The *Compaq C* RTL adds functionality to this original specification. Since, as implemented in the *Compaq C* RTL, the Standard I/O functions recognize line terminators, the *Compaq C* RTL Standard I/O functions are particularly useful for text manipulation. The *Compaq C* RTL also implements some of the Standard I/O functions as preprocessor defined macros.

In a similar manner, the UNIX I/O functions originally were designed to provide a more direct access to the UNIX operating systems. These functions were meant to use a numeric file descriptor to represent a file. A UNIX system represents all peripheral devices as files to provide a uniform method of access.

The *Compaq C* RTL adds functionality to the original specification. The UNIX I/O functions, as implemented in *Compaq C*, are particularly useful for manipulating binary data. The *Compaq C* RTL also implements some of the UNIX I/O functions as preprocessor defined macros.

The *Compaq C* RTL includes the Standard I/O functions that should exist on all C compilers, and also the UNIX I/O functions to maintain compatibility with as many other implementations of C as possible. However, both Standard I/O and UNIX I/O use RMS to access files. To understand how the Standard I/O and UNIX I/O functions manipulate RMS formatted files, learn the fundamentals of RMS. See Section 1.8.1 for more information about Standard I/O and UNIX I/O in relationship to RMS files. For an introduction to RMS, see the *Guide to OpenVMS File Applications*.

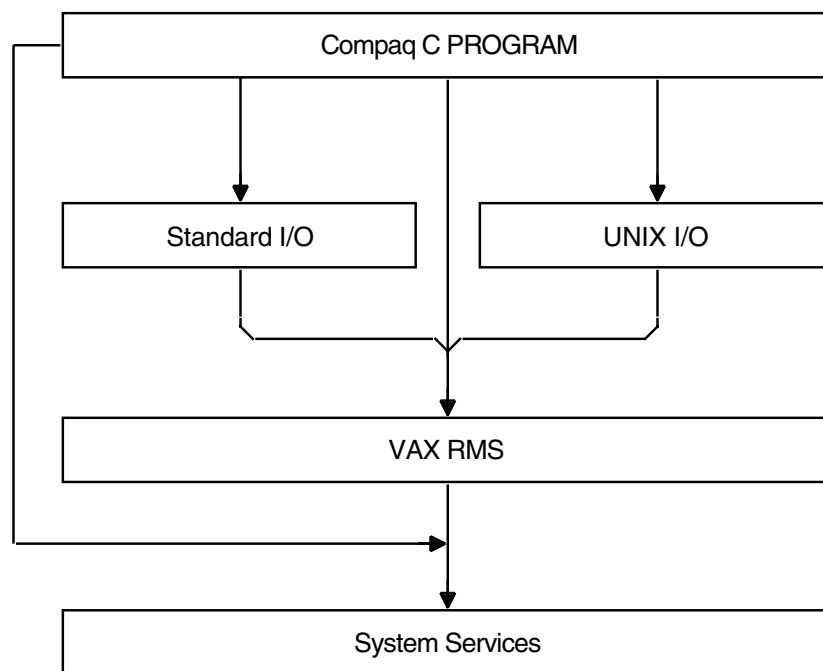
Before deciding which method is appropriate for you, first ask this question: Are you concerned with UNIX compatibility or with developing code that will run solely under the *OpenVMS* operating system?

- If UNIX compatibility is important, you probably want to use the highest levels of I/O—Standard I/O and UNIX I/O—because that level is largely independent of the operating system. Also, the highest level is easier to learn quickly, an important consideration if you are a new programmer.
- If UNIX compatibility is not important to you or if you require the sophisticated file processing that the Standard I/O and UNIX I/O methods do not provide, you might find RMS desirable.

If you are writing system-level software, you may need to access the *OpenVMS* operating system directly through calls to system services. For example, you may need to access a user-written device driver directly through the Queue I/O Request System Service (\$QIO). To do this, use the *OpenVMS* level of I/O; this level is recommended if you are an experienced *OpenVMS* programmer. For examples of programs that call *OpenVMS* system services, see the *Compaq C User's Guide for OpenVMS Systems*.

You may never use the RMS or the *OpenVMS* system services. The Standard I/O and UNIX I/O functions are efficient enough for a large number of applications. Figure 1–3 shows the dependency of the Standard I/O and the UNIX I/O functions on RMS, and the various methods of I/O available to you.

Figure 1–3 Mapping Standard I/O and UNIX I/O to RMS



ZK-0494-GE

1.8.1 RMS Record and File Formats

To understand the capabilities and the restrictions of the Standard I/O and UNIX I/O functions and macros, you need to understand VAX Record Management Services (RMS).

RMS supports the following file organizations:

- Sequential
- Relative
- Indexed

Sequential files have consecutive records with no empty records in between; relative files have fixed-length cells that may or may not contain a record; and indexed files have records that contain data, carriage-control information, and keys that permit various orders of access.

The *Compaq C* RTL functions can access only sequential files. If you wish to use the other file organizations, you must use the RMS functions. For more information about the RMS functions, see the *Compaq C User's Guide for OpenVMS Systems*.

RMS is not concerned with the contents of records, but it is concerned about the record format, which is the way a record physically appears on the recording surface of the storage medium.

RMS supports the following record formats:

- Fixed-length
- Variable-length
- Variable with fixed-length control (VFC)
- Stream

You can specify a fixed-length record format at the time of file creation. This means that all records occupy the same amount of space in the file. You cannot change the record format once you create the file.

The length of records in variable-length, VFC, and stream file formats can vary up to a maximum size that must be specified when you create the file. With variable-length record or VFC format files, the size of the record is held in a header section at the beginning of the data record. With stream files, RMS terminates the records when it encounters a specific character, such as a carriage-control or line-feed character. Stream files are useful for storing text.

RMS allows you to specify carriage-control attributes for records in a file. Such attributes include the implied carriage-return or the Fortran formatted records. RMS interprets these carriage controls when the file is output to a terminal, a line printer, or other device. The carriage-control information is not stored in the data records.

By default, files inherit the RMS record format, maximum record size and record attributes, from the previous version of the file, if one exists; to an *OpenVMS* system programmer, the inherited attributes are known as `fab$b_rfm`, `fab$w_mrs` and `fab$b_rat`. If no previous versions exist, the newly created file defaults to stream format with line-feed record separator and implied carriage-return attributes. (This manual refers to this type of file as a *stream file*.) You can manipulate stream files using the Standard I/O and the UNIX I/O functions of the *Compaq C* RTL. When using these files and fixed-record files with no carriage control, there is no restriction on the ability to seek to any random byte of the file using the `fseek` or the `lseek` functions. However, if the file has one of the other RMS record formats, such as variable-length record format, then these functions, due to RMS restrictions, can seek only to record boundaries. Use the default VAX stream format unless you need to create or access files to be used with other VAX languages or utilities.

1.8.2 Access to RMS Files

RMS sequential files can be opened in record mode or stream mode. By default, `STREAM_LF` files are opened in stream mode; all other file types are opened in record mode. When opening a file, you can override these defaults by specifying the optional argument `"ctx=rec"` to force record mode, or `"ctx=stm"` to force stream mode. RMS relative and indexed files are always opened in record mode. The access mode determines the behavior of various I/O functions in the *Compaq C* RTL.

One of the file types defined by RMS is an RMS-11 stream format file, corresponding to a value of FAB\$C_STM for the record format. The definition of this format is such that the RMS record operation SYS\$GET removes leading null bytes from each record. Because this file type is processed in record mode by the *Compaq C* RTL, it is unsuitable as a file format for binary data unless it is explicitly opened with "ctx=stm", in which case the raw bytes of data from the file are returned.

Note

In *OpenVMS* Version 7.0 the default LRL value on stream files was changed from 0 to 32767. This change caused significant performance degradation on certain file operations such as sort.

This is no longer a problem. The *Compaq C* RTL now lets you define the logical DECC\$DEFAULT_LRL to change the default record-length value on stream files.

The *Compaq C* RTL first looks for this logical. If it is found and it translates to a numeric value between 0 and 32767, that value is used for the default LRL.

To restore the behavior prior to *OpenVMS* Version 7.0, enter the following command:

```
$ DEFINE DECC$DEFAULT_LRL 0
```

1.8.2.1 Accessing RMS Files in Stream Mode

Stream access to RMS files is done with the block I/O facilities of RMS. Stream input is performed from RMS files by passing each byte of the on-disk representation of the file to your program. Stream output to RMS files is done by passing each byte from your program to the file. The *Compaq C* RTL performs no special processing on the data.

When opening a file in stream mode, the *Compaq C* RTL allocates a large internal buffer area. Data is read from the file using a single read into the buffer area and then passing the data to your program as needed. Data is written to the file when the internal buffer is full or when the `fflush` function is called.

1.8.2.2 Accessing RMS Record Files in Record Mode

Record access to record files is done with the record I/O facilities of RMS. The *Compaq C* RTL emulates a byte stream by translating carriage-control characters during the process of reading and writing records. Random access is allowed to all record files, but positioning (with `fseek` and `lseek`) must be on a record boundary for VFC files, variable record files, or files with non-null carriage control. Positioning a record file causes all buffered input to be discarded and buffered output to be written to the file.

Record input from RMS record files is emulated by the *Compaq C* RTL in two steps:

1. The *Compaq C* RTL reads a logical record from the file.

If the record format is variable length with fixed control (RFM = VFC), and the record attributes are not print carriage control (RAT is not PRN), then the *Compaq C* RTL concatenates the fixed-control area to the beginning of the record.

2. The *Compaq C* RTL expands the record to simulate a stream of bytes by translating the record's carriage-control information (if any).

In RMS terms, the *Compaq C* RTL translates the record's carriage-control information using one of the following methods:

- If the record attribute is implied carriage control (RAT = CR), then the *Compaq C* RTL appends a new-line character to the record.

This new-line character is considered an integral part of the record, which means for example, that it can be obtained by the `fgetc` function and is considered a line terminator by the `fgets` function. Since `fgets` reads the file up to the new-line character, for RAT=CR files this function cannot retrieve a string that crosses the record boundaries.

- If the record attributes are print carriage control (RAT = PRN), then the *Compaq C* RTL expands and concatenates the prefix and postfix carriage controls before and after the record.

This translation is done according to rules specified by RMS, with one exception: if the prefix character is `x01` and the postfix character is `x8D`, then nothing is attached to the beginning of the record and a single new-line character is attached to the end of it. This is done because this prefix/postfix combination is normally used to represent a line.

- If the record attributes are Fortran carriage control (RAT = FTN), then the *Compaq C* RTL removes the initial control byte and attaches the appropriate carriage-control characters before and after the data as defined by RMS, with the exception of the space and default carriage-control characters. In these cases, which are used to represent a line, the *Compaq C* RTL appends a single new-line character to the data.

The mapping of Fortran carriage-control can be disabled by using `"ctx=nocvt"`.

- If the record attributes are null (RAT = NONE) and the input is coming from a terminal, then the *Compaq C* RTL appends the terminating character to the record. If the terminator is a carriage return or `Ctrl/Z`, then *Compaq C* translates the character to a new-line character (`\n`).

If the input is coming from a nonterminal file, then the *Compaq C* RTL passes the record unchanged to your program with no additional prefix or postfix characters.

As you read from the file, the *Compaq C* RTL delivers a stream of bytes resulting from the translations. Information that is not read from an expanded record by one function call is delivered on the next input function call.

The *Compaq C* RTL performs record output to RMS record files in two steps.

The first part of the record output emulation is the formation of a logical record. As you write bytes to a record file, the emulator examines the information being written for record boundaries. The handling of information in the byte stream depends on the attributes of the destination file or device, as follows:

- For all files, if the number of output bytes is greater than the internal buffer allocated by the *Compaq C* RTL, a record is output.
- For files with fixed record length (RFM = FIX) or for files opened with `"ctx=bin"` or `"ctx=xplt"`, a record is output only when the internal buffer is filled or when the flush function is called.
- For files with STREAM_CR record format (RFM = STMCR), the *Compaq C* RTL outputs a record when it encounters a carriage-return character (`\r`).

- For files with STREAM record format (RFM = STM) the *Compaq C* RTL outputs a record when it encounters a new-line (\n), form feed (\f), or vertical tab (\v) character.
- For all other file types, the *Compaq C* RTL outputs a record when it encounters a new-line (\n) character.

The second part of record output emulation is to write the logical record formed during the first step. The *Compaq C* RTL forms the output record as follows:

- If the record attribute is carriage control (R AT = CR), and if the logical record ends with a new-line character (\n), the *Compaq C* RTL drops the new-line character and writes the logical record with implied carriage control.
- If the record attribute is print carriage control (RAT = PRN), then the *Compaq C* RTL writes the record with print carriage control according to the rules specified by RMS. If the logical record ends with a single new-line character (\n), the *Compaq C* RTL maps the new-line character to an x01 prefix and x8D postfix character. This is the reverse of the translation for record input files with print carriage-control attributes.
- If the record attributes are Fortran carriage control (RAT = FTN), then the *Compaq C* RTL removes any prefix and/or postfix carriage control characters and concatenates a single carriage-control byte to the beginning of the record as defined by RMS, with one exception: If the output record ends in a new-line character (\n), the *Compaq C* RTL will remove the new-line and use the space carriage-control byte. This is the reverse of the translation for record input files with Fortran carriage-control attributes.

The mapping of Fortran carriage-control can be disabled by using "ctx=nocvt".

- If the logical record is to be written to a terminal device and the last character of the record is a new-line character (\n) the *Compaq C* RTL replaces the new-line character with a carriage-return (\r), and attaches a line-feed character (\n) to the front of the record. The *Compaq C* RTL then writes out the record with no carriage control.
- If the output file record format is variable length with fixed control (RFM = VFC), and the record attributes do not include print carriage control (RAT is not PRN), then the *Compaq C* RTL takes the beginning of the logical record to be the fixed-control header, and reduces the number of bytes written out by the length of the header. These bytes are then used to construct the fixed-control header. If there are too few bytes in the logical record, an error is signaled.

1.8.2.2.1 Accessing Variable-Length or VFC Record Files in Record Mode

When you access a variable-length or VFC record file in record mode, many I/O functions behave differently than they would if they were being used with stream mode. This section describes these differences.

In general, the new-line character (\n) is the record separator for all record modes. On output, when a new-line character is encountered, a record is generated unless you specify an optional argument (such as "ctx=bin" or "ctx=xplt") that affects the interpretation of new lines.

The read and decc\$record_read functions always read at most one record. The write and decc\$record_write functions always generate at least one record.

decc\$record_read and decc\$record_write are equivalent, respectively, to read and write, except that they work with file pointers rather than file descriptors.

Unlike the read function, which reads at most one record, the fread function can span records. Rather than read *number_items* records (where *number_items* is the third parameter to fread), fread tries to read the number of bytes equal to *number_items* × *size_of_item* (where *size_of_item* is the second parameter to fread). The value returned by fread is equal to the number of bytes read divided by *size_of_item*.

However, the fwrite function always generates at least *number_items* records.

The fgets and gets functions read to either a new-line character or a record boundary.

The fflush function always generates a record if there is unwritten data in the buffer. The same is true of close, fclose, fseek, lseek, rewind, and fsetpos, all of which perform implicit fflush functions.

A record is also generated whenever an attempt is made to write more characters than allowed by the maximum record size.

For more information on these functions, see the Reference Section.

1.8.2.2.2 Accessing Fixed-Length Record Files in Record Mode When accessing a fixed-length record file in record mode, the I/O functions generally behave as described in Section 1.8.2.2.1.

The write, fwrite, and decc\$record_write functions will fail if given a record size that is not an integral multiple of the maximum record size, unless the file was opened with the "ctx=xplct" optional argument specified. All other output functions will generate records at every *n*th byte, where *n* is the maximum record size.

If a new record is forced by fflush, the data in the buffer is padded to the maximum record size with null characters.

Note

This padding can cause problems for programs that seek to the end-of-file. For example, if a program were to append data to a file, then seek backwards in the file (causing an fflush to occur), and then seek to the end-of-file again, a zero-filled "hole" will have been created between the previous end-of-file and the new end-of-file if the previous end-of-file was not on a record boundary.

1.8.2.3 Example—Difference Between Stream Mode and Record Mode

Example 1–1 demonstrates the difference between stream mode and record mode access.

Example 1–1 Differences Between Stream Mode and Record Mode Access

```
/*      CHAP_1_STREAM_RECORD.C      */
/* This program demonstrates the difference between */
/* record mode and stream mode input/output.      */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

(continued on next page)

Example 1–1 (Cont.) Differences Between Stream Mode and Record Mode Access

```
void process_records(const char *fspec, FILE * fp);

main()
{
    FILE *fp;

    fp = fopen("example-fixed.dat", "w", "rfm=fix", "mrs=40", "rat=none");
    if (fp == NULL) {
        perror("example-fixed");
        exit(EXIT_FAILURE);
    }
    printf("Record mode\n");
    process_records("example-fixed.dat", fp);
    fclose(fp);

    printf("\nStream mode\n");
    fp = fopen("example-streamlf.dat", "w");
    if (fp == NULL) {
        perror("example-streamlf");
        exit(EXIT_FAILURE);
    }
    process_records("example-streamlf.dat", fp);
    fclose(fp);
}

void process_records(const char *fspec, FILE * fp)
{
    int i,
        sts;

    char buffer[40];

    /* Write records of all 1's, all 2's and all 3's */
    for (i = 0; i < 3; i++) {
        memset(buffer, '1' + i, 40);
        sts = fwrite(buffer, 40, 1, fp);
        if (sts != 1) {
            perror("fwrite");
            exit(EXIT_FAILURE);
        }
    }

    /* Rewind the file and write 10 characters of A's, then 10 B's, */
    /* then 10 C's. */
    /*
    /* For stream mode, each fwrite call outputs 10 characters
    /* and advances the file position 10 characters
    /* characters.
    /*
    /* For record mode, each fwrite merges the 10 characters into
    /* the existing 40-character record, updates the record and
    /* advances the file position 40 characters to the next record. */
    rewind(fp);
    for (i = 0; i < 3; i++) {
        memset(buffer, 'A' + i, 10);
        sts = fwrite(buffer, 10, 1, fp);
        if (sts != 1) {
            perror("fwrite2");
            exit(EXIT_FAILURE);
        }
    }
}
```

(continued on next page)


```
/* Now reopen the file and output the records. */
fclose(fp);
fp = fopen(fspect, "r");
for (i = 0; i < 3; i++) {
    sts = fread(buffer, 40, 1, fp);
    if (sts != 1)
        perror("fread");
    printf("%.40s\n", buffer);
}

return;
}
```

[illegible]

One of the last tasks in preparing to use the *Compaq C* RTL, if you are going to port your source programs across systems, is to be aware of specific differences between the *Compaq C* RTL and the run-time libraries of other implementations of the C language. This section describes some of the problems that you might encounter when porting programs to and from an *OpenVMS* system. Although portability is closely tied to the implementation of the *Compaq C* RTL, this section also contains information on the portability of other *Compaq C* for *OpenVMS* constructs.

The following list documents issues of concern if you wish to port C programs to the *OpenVMS* environment:

- ## 1-44 Introduction

- There are differences in how *OpenVMS* and UNIX systems lay out virtual memory. In some UNIX systems, the address space between 0 and the break address is accessible to your program. In *OpenVMS* systems, the first page of memory is not accessible.

For example, if a program tries to reference location 0 on an *OpenVMS* system, a hardware error (ACCVIO) is returned and the program terminates abnormally. *OpenVMS* systems reserve the first page of address space to catch incorrect pointer references, such as a reference to a location pointed to by a null pointer. For this reason, some existing programs that run on some UNIX systems may fail and you should modify them, as necessary. (*Tru64 UNIX* and *OpenVMS*, however, are compatible in this regard.)

- Some C programmers code all external declarations in `#include` files. Then, specific declarations that require initialization are redeclared in the relevant module. This practice causes the *Compaq C* compiler to issue a warning message about multiply declared variables in the same compilation. One way to avoid this warning is to make the redeclared symbols extern variables in the `#include` files.
- *Compaq C* does not support `asm` calls on *OpenVMS* VAX systems. They are supported on *OpenVMS* Alpha systems. See the *Compaq C User's Guide for OpenVMS Systems* for more information on intrinsic functions.
- Some C programs call the counted string functions `strcmprn` and `strcpyn`. These names are not used by *Compaq C* for *OpenVMS* Systems. Instead, you can define macros that expand the `strcmprn` and `strcpyn` names into the equivalent, ANSI-compliant names `strncmp` and `strncpy`.
- The *Compaq C* for *OpenVMS* compiler does not support the following initialization form:

```
int foo 123;
```

Programs using this form of initialization must be changed.

- *Compaq C* for *OpenVMS* Systems predefines several compile-time macros such as `__vax`, `__alpha`, `__32BITS`, `__vms`, `__vaxc`, `__VMS_VER`, `__DECC_VER`, `__D_FLOAT`, `__G_FLOAT`, `__IEEE_FLOAT`, `__X_FLOAT`, and others. These predefined macros are useful for programs that must be compatible on other machines and operating systems. For more information, see the predefined macro chapter of the *Compaq C User's Guide for OpenVMS Systems*.
 - The ANSI C language does not guarantee any memory order for the variables in a declaration. For example:
- ```
int a, b, c;
```
- Depending on the type of external linkage requested, extern variables in a program may be treated differently using *Compaq C* on *OpenVMS* systems than they would on UNIX systems. See the *Compaq C User's Guide for OpenVMS Systems* for more information.
  - The dollar sign (\$) is a legal character in *Compaq C* for *OpenVMS* identifiers, and can be used as the first character.
  - The ANSI C language does not define any order for evaluating expressions in function parameter lists or for many kinds of expressions. The way in which different C compilers evaluate an expression is only important when the expression has side effects. Consider the following examples:

```

a[i] = i++;
x = func_y() + func_z();
f(p++, p++)

```

Neither *Compaq C* nor any other C compiler can guarantee that such expressions evaluate in the same order on all C compilers.

- The size of a *Compaq C* variable of type `int` is 32 bits on *OpenVMS* systems. You will have to modify programs that are written for other machines and that assume a different size for a variable of type `int`. A variable of type `long` is the same size (32 bits) as a variable of type `int`.
- The C language defines structure alignment to be dependent on the machine for which the compiler is designed. On *OpenVMS VAX* systems, *Compaq C* aligns structure members on byte boundaries, unless `#pragma member_alignment` is specified. On *OpenVMS Alpha* systems, *Compaq C* aligns structure members on natural boundaries, unless `#pragma nomember_alignment` is specified. Other implementations may align structure members differently.
- References to structure members in *Compaq C* cannot be vague. For more information, see the *Compaq C Language Reference Manual*.
- Registers are allocated based upon how often a variable is used, but the `register` keyword gives the compiler a strong hint that you want to place a particular variable into a register. Whenever possible, the variable is placed into a register. Any scalar variable with the storage class `auto` or `register` can be allocated to a register as long as the variable's address is not taken with the ampersand operator (`&`) and it is not a member of a structure or union.

### 1.9.1 Reentrancy

The *Compaq C* RTL supports an improved and enhanced reentrancy. The following types of reentrancy are supported:

- **AST** reentrancy uses the `_BBSSI` built-in function to perform simple locking around critical sections of RTL code, but it may also disable asynchronous system traps (ASTs) in locked regions of code. This type of locking should be used when AST code contains calls to *Compaq C* RTL I/O routines.  
Failure to specify AST reentrancy might cause I/O routines to fail, setting `errno` to `EALREADY`.
- **MULTITHREAD** reentrancy is designed to be used in threaded programs such as those that use the `DECthreads` library. It performs `DECthreads` locking and never disables ASTs. `DECthreads` must be available on your system to use this form of reentrancy.
- **TOLERANT** reentrancy uses the `_BBSSI` built-in function to perform simple locking around critical sections of RTL code, but ASTs are not disabled. This type of locking should be used when ASTs are used and must be delivered immediately.
- **NONE** gives optimal performance in the *Compaq C* RTL, but does absolutely no locking around critical sections of RTL code. It should only be used in a single-threaded environment when there is no chance that the thread of execution will be interrupted by an AST that would call the *Compaq C* RTL.

The default reentrancy type is **TOLERANT**.

You can set the reentrancy type by compiling with the `/REENTRANCY` command-line qualifier or by calling the `decc$set_reentrancy` function. This function must be called exclusively from non-AST level.

When programming an application using multiple threads or ASTs, consider three classes of functions:

- Functions with no internal data
- Functions with thread-local internal data
- Functions with process-wide internal data

Most functions have no internal data at all. For these functions, synchronization is necessary only if the parameter is used by the application in multiple threads or in both AST and non-AST contexts. For example, although the `strcat` function is ordinarily safe, the following is an example of unsafe usage:

```
extern char buffer[100];
void routine1(char *data) {
 strcat(buffer, data);
}
```

If `routine1` executed concurrently in multiple threads, or if `routine1` is interrupted by an AST routine that calls it, the results of the `strcat` call are unpredictable.

The second class of functions are those that have thread-local static data. Typically, these are routines in the library that return a string where the application is not permitted to free the storage for the string. These routines are thread-safe but not AST-reentrant. This means they can safely be called concurrently, and each thread will have its own copy of the data. They cannot be called from AST routines if it is possible that the same routine was executing in non-AST context. The routines in this class are:

|                        |                                 |
|------------------------|---------------------------------|
| <code>asctime</code>   | <code>stat</code>               |
| <code>ctermid</code>   | <code>strerror</code>           |
| <code>ctime</code>     | <code>strtok</code>             |
| <code>cuserid</code>   | <code>VAXC\$ESTABLISH</code>    |
| <code>gmtime</code>    | the <code>errno</code> variable |
| <code>localtime</code> | <code>wcstok</code>             |
| <code>perror</code>    |                                 |

All the socket functions are also included in this list if the TCP/IP product in use is thread-safe.

The third class of functions are those that affect process-wide data. These functions are neither thread-safe nor AST-reentrant. For example, `sigsetmask` establishes the process-wide signal mask. Consider a routine like the following:

```
void update_data
base()
{
 int old_mask;
 old_mask = sigsetmask(1 << (SIGINT - 1));
 /* Do work here that should not be aborted. */
 sigsetmask(old_mask);
}
```

If `update_data` was called concurrently in multiple threads, thread 1 might unblock `SIGINT` while thread 2 was still performing work that should not be aborted.

The routines in this class are:

- All the signal routines
- All the exec routines
- The `exit`, `_exit`, `nice`, `system`, `wait`, `getitimer`, `setitimer`, and `setlocale` routines.

---

**Note**

---

Generally speaking, UTC-based time functions can affect in-memory time-zone information, which is process-wide data. However, if the system time zone remains the same during the execution of the application (which is the common case) and the cache of time-zone files is enabled (which is the default), then the `_r` variant of the time functions `asctime_r`, `ctime_r`, `gmtime_r` and `localtime_r` is both thread-safe and AST-reentrant.

If, however, the system time zone can change during the execution of the application or the cache of time-zone files is not enabled, then both variants of the UTC-based time functions belong to the third class of functions, which are neither thread-safe nor AST-reentrant.

---

### 1.9.2 Multithread Restrictions

Mixing the multithread programming model and the *OpenVMS* AST programming model in the same application is not recommended. The application has no mechanism to control which thread gets interrupted by an AST. This can result in a resource deadlock if the thread holds a resource that is also needed by the AST routine. The following routines use mutexes. To avoid a potential resource deadlock, do not call them from AST routines in a multithreaded application.

- All the I/O routines
- All the socket routines
- All the signal routines
- `vfork`, `exec`, `wait`, `system`
- `catgets`
- `set_new_handler` (C++ only)
- `getenv`
- `rand` and `srand`
- `exit` and `_exit`
- `clock`
- `nice`
- `times`
- `ctime`, `localtime`, `asctime`, `mktime`

## 1.10 64-bit Pointer Support (Alpha only)

This section is for application developers who need to use 64-bit virtual memory addressing on *OpenVMS Alpha* Version 7.0 or higher.

*OpenVMS Alpha* 64-bit virtual addressing support makes the 64-bit virtual address space defined by the Alpha architecture available to both the *OpenVMS* operating system and its users. It also allows per-process virtual addressing for accessing dynamically mapped data beyond traditional 32-bit limits.

The *Compaq C* Run-Time Library on *OpenVMS Alpha* Version 7.0 systems and higher includes the following features in support of 64-bit pointers:

- Guaranteed binary and source compatibility of existing programs
- No impact on applications that are not modified to exploit 64-bit support
- Enhanced memory allocation routines that allocate 64-bit memory
- Widened function parameters to accommodate 64-bit pointers
- Dual implementations of functions that need to know the pointer size used by the caller
- New information available to the DEC C Version 5.2 compiler or higher to seamlessly call the correct implementation
- Ability to explicitly call either the 32-bit or 64-bit form of functions for applications that mix pointer sizes
- A single shareable image for use by 32-bit and 64-bit applications

### 1.10.1 Using the *Compaq C* Run-Time Library

The *Compaq C* Run-Time library on *OpenVMS Alpha* Version 7.0 systems and higher can generate and accept 64-bit pointers. Functions that require a second interface to be used with 64-bit pointers reside in the same object libraries and shareable images as their 32-bit counterparts. No new object libraries or shareable images are introduced. Using 64-bit pointers does not require changes to your link command or link options files.

The *Compaq C* 64-bit environment allows an application to use both 32-bit and 64-bit addresses. For more information about how to manipulate pointer sizes, see the `/POINTER_SIZE` qualifier and `#pragma pointer_size` and `#pragma required_pointer_size` preprocessor directives in the *Compaq C User's Guide for OpenVMS Systems*.

The `/POINTER_SIZE` qualifier requires you to specify a value of 32 or 64. This value is used as the default pointer size within the compilation unit. As an application programmer, you can compile one set of modules by using 32-bit pointers and another set by using 64-bit pointers. Care must be taken when these two separate groups of modules call each other.

Use of the `/POINTER_SIZE` qualifier also influences the processing of *Compaq C* RTL header files. For those functions that have a 32-bit and 64-bit implementation, specifying `/POINTER_SIZE` enables function prototypes to access both functions, regardless of the actual value supplied to the qualifier. In addition, the value specified to the qualifier determines the default implementation to call during that compilation unit.

The `#pragma pointer_size` and `#pragma required_pointer_size` preprocessor directives can be used to change the pointer size in effect within a compilation unit. You can default pointers to 32-bit pointers and then declare specific pointers within the module as 64-bit pointers. You would also need to specifically call the `_malloc64` form of `malloc` to obtain memory from the 64-bit memory area.

### 1.10.2 Obtaining 64-bit Pointers to Memory

The *Compaq C* RTL has many functions that return pointers to newly allocated memory. In each of these functions, the application owns the memory pointed to and is responsible for freeing that memory.

Functions that allocate memory are:

```
malloc
calloc
realloc
strdup
```

Each of these functions have a 32-bit and a 64-bit implementation. When the `/POINTER_SIZE` qualifier is used, the following functions can also be called:

```
_malloc32, _malloc64
_calloc32, _calloc64
_realloc32, _realloc64
_strdup32, _strdup64
```

When `/POINTER_SIZE=32` is specified, all `malloc` calls default to `_malloc32`.

When `/POINTER_SIZE=64` is specified, all `malloc` calls default to `_malloc64`.

Regardless of whether the application calls a 32-bit or 64-bit memory allocation routine, there is still a single `free` function. This function accepts either pointer size.

Be aware that the memory allocation functions are the only ones that return pointers to 64-bit memory. All *Compaq C* RTL structure pointers returned to the calling application (such as a `FILE`, `WINDOW`, or `DIR`) are always 32-bit pointers. This allows both 32-bit and 64-bit callers to pass these structure pointers within the application.

### 1.10.3 *Compaq C* Header Files

The header files distributed with DEC C Version 5.2 and higher support 64-bit pointers. Each function prototype whose signature contains a pointer is constructed to indicate the size of the pointer accepted.

A 32-bit pointer can be passed as an argument to functions that accept either a 32-bit or 64-bit pointer for that argument.

A 64-bit pointer, however, cannot be passed as an argument to a function that accepts a 32-bit pointer. Attempts to do this are diagnosed by the compiler with a `MAYLOSEDATA` message. The diagnostic message `IMPLICITFUNC` means the compiler can do no additional pointer-size validation for calls to that function. If this function is a *Compaq C* RTL function, refer to the reference section of this manual for the name of the header file that defines that function.

You might find the following pointer-size compiler diagnostics useful:

- `%CC-IMPLICITFUNC`

A function prototype was not found before using the specified function. The compiler and run-time system rely on prototype definitions to detect incorrect pointer-size usage. Failure to include the proper header files can lead to incorrect results and/or pointer truncation.

- **%CC-MAYLOSEDATA**

A truncation is necessary to do this operation. The operation could be passing a 64-bit pointer to a function that does not support a 64-bit pointer in the given context. It could also be a function returning a 64-bit pointer to a calling application that is trying to store that return value in a 32-bit pointer.

- **%CC-MAYHIDELOSS**

This message (when enabled) helps expose real MAYLOSEDATA messages that are being suppressed because of a cast operation. To enable this warning, compile with the qualifier /WARNINGS=ENABLE=MAYHIDELOSS.

### 1.10.4 Functions Affected

The *Compaq C* RTL shipped with *OpenVMS Alpha* Version 7.0 accommodates applications that use only 32-bit pointers, only 64-bit pointers, or combinations of both. To use 64-bit memory, you must, at a minimum, recompile and relink an application. The amount of source code change required depends on the application itself, calls to other run-time libraries, and the combinations of pointer sizes used.

With respect to 64-bit pointer support, the functions in the *Compaq C* RTL fall into four categories:

- Functions not impacted by choice of pointer size
- Functions enhanced to accept either pointer size
- Functions having a 32-bit and 64-bit implementation
- Functions that accept only 32-bit pointers

From an application developer's perspective, the first two types of functions are the easiest to use in either a single- or mixed-pointer mode.

The third type requires no modifications when used in a single-pointer compilation, but might require source code changes when used in a mixed-pointer mode.

The fourth type requires careful attention whenever 64-bit pointers are used.

#### 1.10.4.1 No Pointer-Size Impact

The choice of pointer size has no impact on a function if its prototype contains no pointer-related parameters or return values. The mathematical functions are good examples of this.

Even some functions in this category that do have pointers in their prototype are not impacted by pointer size. For example, `strerror` has the prototype:

```
char * strerror (int error_number);
```

This function returns a pointer to a character string, but this string is allocated by the *Compaq C* RTL. As a result, to support both 32-bit and 64-bit applications, these types of pointers are guaranteed to fit in a 32-bit pointer.



#### 1.10.4.2 Functions Accepting Both Pointer Sizes

The Alpha architecture supports 64-bit pointers. The *OpenVMS Alpha* calling standard specifies that all arguments are actually passed as 64-bit values. Before *OpenVMS Alpha* Version 7.0, all 32-bit addresses passed to procedures were sign-extended into this 64-bit parameter. The called function declared the parameters as 32-bit addresses, which caused the compiler to generate 32-bit instructions (such as LDL) to manipulate these parameters.

Many functions in the *Compaq C* RTL are enhanced to receive the full 64-bit address. For example, consider `strlen`:

```
size_t strlen (const char *string);
```

The only pointer in this function is the character-string pointer. If the caller passes a 32-bit pointer, the function works with the sign-extended 64-bit address. If the caller passes a 64-bit address, the function works directly with that address.

The *Compaq C* RTL continues to have only a single entry point for functions in this category. There are no source-code changes required to add any of the four pointer-size options for functions of this type. The *OpenVMS* documentation refers to these functions as 64-bit friendly.

#### 1.10.4.3 Functions with Two Implementations

There are many reasons why a function might need one implementation for 32-bit pointers and another for 64-bit pointers. Some of these reasons include:

- The pointer size of the return value is the same size as the pointer size of one of the arguments. If the argument is 32 bits, the return value is 32 bits. If the argument is 64 bits, the return value is 64 bits.
- One of the arguments is a pointer to an object whose size is pointer-size sensitive. To know how many bytes are being pointed to, the function must know if the code was compiled in 32-bit or 64-bit pointer-size mode.
- The function returns the address of dynamically allocated memory. The memory is allocated in 32-bit space when compiled for 32-bit pointers, and is allocated in 64-bit space when compiled for 64-bit pointers.

From the application developer's point of view, there are three function prototypes for each of these functions. The `<string.h>` header file contains many functions whose return value is dependent upon the pointer size used as the first argument to the function call. For example, consider the `memset` function. The header file defines three entry points for this function:

```
void * memset (void *memory_pointer, int character, size_t size);
void * memset32 (void *memory_pointer, int character, size_t size);
void * _memset64 (void *memory_pointer, int character, size_t size);
```

The first prototype is the function that your application would currently call if using this function. The compiler changes a call to `memset` into a call to either `_memset32` when compiled `/POINTER_SIZE=32`, or `_memset64` when compiled `/POINTER_SIZE=64`.

You can override this default behavior by directly calling either the 32-bit or the 64-bit form of the function. This accommodates applications using mixed-pointer sizes, regardless of the default pointer size specified with the `/POINTER_SIZE` qualifier.



If the application is compiled *without* specifying the `/POINTER_SIZE` qualifier, *neither* the 32-bit specific *nor* the 64-bit specific function prototypes are defined. In this case, the compiler automatically calls the 32-bit interface for all interfaces having dual implementations.

Table 1–6 shows the *Compaq C* RTL functions that have dual implementations in support of 64-bit pointer size. When compiling with the `/POINTER_SIZE` qualifier, calls to the unmodified function names are changed to calls to the function interface that matches the pointer size specified on the qualifier.

**Table 1–6 Functions with Dual Implementations**

|          |           |          |           |
|----------|-----------|----------|-----------|
| basename | malloc    | strpbrk  | wcsncat   |
| bsearch  | mbsrtowcs | strptime | wcsncpy   |
| calloc   | memccpy   | strrchr  | wcspbrk   |
| catgets  | memchr    | strsep   | wcsrchr   |
| ctermid  | memcpy    | strstr   | wcsrtombs |
| cuserid  | memmove   | strtod   | wcsstr    |
| dirname  | memset    | strtok   | wcstok    |
| fgetname | mktemp    | strtol   | wcstol    |
| fgets    | mmap      | strtoll  | wcstoul   |
| fgetws   | qsort     | strtoq   | wcswcs    |
| fullname | realloc   | strtoul  | wmemchr   |
| gcvt     | rindex    | strtoull | wmemcpy   |
| getcap   | strcat    | strtouq  | wmemmove  |
| getcwd   | strchr    | tgetstr  | wmemset   |
| getname  | strcpy    | tmpnam   |           |
| gets     | strdup    | wscat    |           |
| index    | strncat   | wcschr   |           |
| longname | strncpy   | wcscpy   |           |

#### 1.10.4.4 Functions Restricted to 32-Bit Pointers

A few functions in the *Compaq C* RTL do not support 64-bit pointers. If you try to pass a 64-bit pointer to one of these functions, the compiler generates a `%CC-W-MAYLOSEDATA` warning. Applications compiled with `/POINTER_SIZE=64` might need to be modified to avoid passing 64-bit pointers to these functions.

Table 1–7 shows the functions restricted to using 32-bit pointers. The *Compaq C* RTL offers no 64-bit support for these functions. You must ensure that only 32-bit pointers are used with these functions.

**Table 1–7 Functions restricted to 32-bit pointers**

|        |           |         |          |
|--------|-----------|---------|----------|
| atexit | frexp     | ioctl   | sendmsg  |
| execv  | getopt    | modf    | setbuf   |
| execve | iconv     | putenv  | setstate |
| execvp | initstate | recvmsg | setvbuf  |

Table 1–8 shows functions that make callbacks to user-supplied functions as part of processing that function call. The callback procedures are not passed 64-bit pointers.

**Table 1–8 Callbacks that Pass Only 32-Bit Pointers**

|                |              |
|----------------|--------------|
| decc\$from_vms | decc\$to_vms |
| ftw            | tputs        |

### 1.10.5 Reading Header Files

This section introduces the pointer-size manipulations used in the *Compaq C* RTL header files. Use the following examples to become more proficient in reading these header files and to help modify your own header files.

#### Examples

```
1. :
 :
 #if __INITIAL_POINTER_SIZE ❶
 # if (__VMS_VER < 70000000) || !defined __ALPHA ❷
 # error "Pointer size usage not permitted before OpenVMS Alpha V7.0"
 # endif
 # pragma __pointer_size __save ❸
 # pragma __pointer_size 32 ❹
 #endif
 :
 :
 #if __INITIAL_POINTER_SIZE ❺
 # pragma __pointer_size 64
 #endif
 :
 :
 #if __INITIAL_POINTER_SIZE ❻
 # pragma __pointer_size __restore
 #endif
 :
```

All *Compaq C* compilers that support the `/POINTER_SIZE` qualifier predefine the macro `__INITIAL_POINTER_SIZE`. The *Compaq C* RTL header files take advantage of the ANSI rule that if a macro is not defined, it has an implicit value of 0.

The macro is defined as 32 or 64 when the `/POINTER_SIZE` qualifier is used. It is defined as 0 if the qualifier is not used. The statement at ❶ can be read as "if the user has specified either `/POINTER_SIZE=32` or `/POINTER_SIZE=64` on the command line".

DEC C Version 5.2 and higher is supported on many *OpenVMS* platforms. The lines at ❷ generate an error message if the target of the compilation is one that does not support 64-bit pointers.

A header file cannot assume anything about the actual pointer-size context in effect at the time the header file is included. Furthermore, the *Compaq C* compiler offers only the `__INITIAL_POINTER_SIZE` macro and a mechanism to change the pointer size, but not a way to determine the current pointer size.

All header files that have a dependency on pointer sizes are responsible for saving ❸, initializing ❹, altering ❺, and restoring ❻ the pointer-size context.

```

2. :
 :
 #ifndef __CHAR_PTR32 ❶
 # define __CHAR_PTR32 1
 # typedef char * __char_ptr32;
 # typedef const char * __const_char_ptr32;
 #endif
 :
 :
 #if __INITIAL_POINTER_SIZE
 # pragma __pointer_size 64
 #endif
 :
 :
 #ifndef __CHAR_PTR64 ❷
 # define __CHAR_PTR64 1
 # typedef char * __char_ptr64;
 # typedef const char * __const_char_ptr64;
 #endif
 :

```

Some function prototypes need to refer to a 32-bit pointer when in a 64-bit pointer-size context. Other function prototypes need to refer to a 64-bit pointer when in a 32-bit pointer-size context.

*Compaq C* binds the pointer size used in a typedef at the time the typedef is made. The typedef declaration of `__char_ptr32` ❶ is made in a 32-bit context. The typedef declaration of `__char_ptr64` ❷ is made in a 64-bit context.

```

3. :
 :
 #if __INITIAL_POINTER_SIZE
 # if (__VMS_VER < 70000000) || !defined __ALPHA
 # error "Pointer size usage not permitted before OpenVMS Alpha V7.0"
 # endif
 # pragma __pointer_size __save
 # pragma __pointer_size 32
 #endif
 :
 :
 ❶
 :
 :
 #if __INITIAL_POINTER_SIZE ❷
 # pragma __pointer_size 64
 #endif
 :
 :
 ❸
 :
 :
 int abs (int __j); ❹
 :
 :
 __char_ptr32 strerror (int __errnum); ❺
 :

```

Before declaring function prototypes that support 64-bit pointers, the pointer context is changed ❷ from 32-bit pointers to 64-bit pointers.

Functions restricted to 32-bit pointers are placed in the 32-bit pointer context section ❶ of the header file. All other functions are placed in the 64-bit context section ❸ of the header file.

Functions that have no pointer-size impact (❹ and ❺) are located in the 64-bit section. Functions that have no pointer-size impact except for a 32-bit address return value ❺ are also in the 64-bit section, and use the 32-bit specific typedefs previously discussed.

```

4. :
 :
 #if __INITIAL_POINTER_SIZE
 # pragma __pointer_size 64
 #endif
 :
 :
 :
 #if __INITIAL_POINTER_SIZE == 32 ❶
 # pragma __pointer_size 32
 #endif
 :
 char *strcat (char *__s1, __const_char_ptr64 __s2); ❷
 :
 #if __INITIAL_POINTER_SIZE
 # pragma __pointer_size 32
 :
 :
 char *_strcat32 (char *__s1, __const_char_ptr64 __s2); ❸
 :
 # pragma __pointer_size 64
 :
 :
 char *_strcat64 (char *__s1, const char *__s2); ❹
 :
 #endif
 :

```

This example shows declarations of functions that have both a 32-bit and 64-bit implementation. These declarations are located in the 64-bit section of the header file.

The normal interface to the function ❷ is declared using the pointer size specified on the `/POINTER_SIZE` qualifier. Because the header file is in 64-bit pointer context and because of the statements at ❶, the declaration at ❷ is made using the same pointer-size context as the `/POINTER_SIZE` qualifier.

The 32-bit specific interface ❸ and the 64-bit specific interface ❹ are declared in 32-bit and 64-bit pointer-size context, respectively.

---

## Understanding Input and Output

There are three types of input and output (I/O) in the *Compaq C* Run-Time Library (RTL): UNIX, Standard, and Terminal. Table 2–1 lists all the I/O functions and macros found in the *Compaq C* RTL. For more detailed information on each function and macro, see the Reference Section.

**Table 2–1 I/O Functions and Macros**

| Function or Macro                         | Description                                                                                                       |
|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>UNIX I/O—Opening and Closing Files</b> |                                                                                                                   |
| close                                     | Closes the file associated with a file descriptor.                                                                |
| creat                                     | Creates a new file.                                                                                               |
| dup, dup2                                 | Allocates a new descriptor that refers to a file specified by a file descriptor returned by open, creat, or pipe. |
| open                                      | Opens a file and positions it at its beginning.                                                                   |
| <b>UNIX I/O—Reading from Files</b>        |                                                                                                                   |
| read                                      | Reads bytes from a file and places them in a buffer.                                                              |
| <b>UNIX I/O—Writing to Files</b>          |                                                                                                                   |
| write                                     | Writes a specified number of bytes from a buffer to a file.                                                       |
| <b>UNIX I/O—Maneuvering in Files</b>      |                                                                                                                   |
| lseek                                     | Positions a stream file to an arbitrary byte position and returns the new position as an int.                     |

(continued on next page)

**Table 2–1 (Cont.) I/O Functions and Macros**

| Function or Macro                                            | Description                                                                                                                                                                   |
|--------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>UNIX I/O—Additional Standard I/O Functions and Macros</b> |                                                                                                                                                                               |
| fstat, stat                                                  | Accesses information about the file descriptor or the file specification.                                                                                                     |
| fsync                                                        | Writes to disk any buffered information for the specified file.                                                                                                               |
| getname                                                      | Returns the file specification associated with a file descriptor.                                                                                                             |
| isapipe                                                      | Returns 1 if the file descriptor is associated with a pipe and 0 if it is not.                                                                                                |
| isatty                                                       | Returns 1 if the specified file descriptor is associated with a terminal and 0 if it is not.                                                                                  |
| lwait                                                        | Waits for completion of pending asynchronous I/O.                                                                                                                             |
| ttyname                                                      | Returns a pointer to the null-terminated name of the terminal device associated with file descriptor 0, the default input device.                                             |
| <b>Standard I/O—Opening and Closing Files</b>                |                                                                                                                                                                               |
| fclose                                                       | Closes a function by flushing any buffers associated with the file control block, and freeing the file control block and buffers previously associated with the file pointer. |
| fdopen                                                       | Associates a file pointer with a file descriptor returned by an open, creat, dup, dup2, or pipe function.                                                                     |
| fopen                                                        | Opens a file by returning the address of a FILE structure.                                                                                                                    |
| freopen                                                      | Substitutes the file, named by a file specification, for the open file addressed by a file pointer.                                                                           |
| <b>Standard I/O—Reading from Files</b>                       |                                                                                                                                                                               |
| fgetc, getc, fgetwc, getw, getwc                             | Returns characters from a specified file.                                                                                                                                     |
| fgets, fgetws                                                | Reads a line from a specified file and stores the string in an argument.                                                                                                      |
| fread                                                        | Reads a specified number of items from a file.                                                                                                                                |
| fscanf, fwscanf, vfscanf, vfwscanf                           | Performs formatted input from a specified file.                                                                                                                               |
| sscanf, swscanf, vsscanf, vswscanf                           | Performs formatted input from a character string in memory.                                                                                                                   |
| ungetc, ungetwc                                              | Pushes back a character into the input stream and leaves the stream positioned before the character.                                                                          |

(continued on next page)

**Table 2–1 (Cont.) I/O Functions and Macros**

| Function or Macro                                                | Description                                                                                             |
|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>Standard I/O—Writing to Files</b>                             |                                                                                                         |
| <code>fprintf, fwprintf, vfprintf, vfwprintf</code>              | Performs formatted output to a specified file.                                                          |
| <code>fputc, putc, putw, putwc, fputwc</code>                    | Writes characters to a specified file.                                                                  |
| <code>fputs, fputws</code>                                       | Writes a character string to a file without copying the string's null terminator.                       |
| <code>fwrite</code>                                              | Writes a specified number of items to a file.                                                           |
| <code>sprintf, swprintf, vsprintf, vswprintf</code>              | Performs formatted output to a string in memory.                                                        |
| <b>Standard I/O—Maneuvering in Files</b>                         |                                                                                                         |
| <code>fflush</code>                                              | Sends any buffered information for the specified file to RMS.                                           |
| <code>fgetpos</code>                                             | Stores the current value of the file position indicator for the stream.                                 |
| <code>fsetpos</code>                                             | Sets the file position indicator for the stream according to the value of the object pointed to.        |
| <code>fseek, fseeko</code>                                       | Positions the file to the specified byte offset in the file.                                            |
| <code>ftell, ftello</code>                                       | Returns the current byte offset to the specified stream file.                                           |
| <code>rewind</code>                                              | Sets the file to its beginning.                                                                         |
| <b>Standard I/O—Additional Standard I/O Functions and Macros</b> |                                                                                                         |
| <code>access</code>                                              | Checks a file to see whether a specified access mode is allowed.                                        |
| <code>clearerr</code>                                            | Resets the error and end-of-file indications for a file.                                                |
| <code>feof</code>                                                | Tests a file to see if the end-of-file has been reached.                                                |
| <code>ferror</code>                                              | Returns a nonzero integer if an error has occurred while reading or writing a file.                     |
| <code>fgetname</code>                                            | Returns the file specification associated with a file pointer.                                          |
| <code>fileno</code>                                              | Returns an integer file descriptor that identifies the specified file.                                  |
| <code>ftruncate</code>                                           | Truncates a file at the specified position.                                                             |
| <code>fwait</code>                                               | Waits for completion of pending asynchronous I/O.                                                       |
| <code>fwide</code>                                               | Sets the orientation a stream.                                                                          |
| <code>mktemp</code>                                              | Creates a unique file name from a template.                                                             |
| <code>remove, delete</code>                                      | Deletes a file.                                                                                         |
| <code>rename</code>                                              | Gives a new name to an existing file.                                                                   |
| <code>setbuf, setvbuf</code>                                     | Associates a buffer with an input or output file.                                                       |
| <code>tmpfile</code>                                             | Creates a temporary file that is opened for update.                                                     |
| <code>tmpnam</code>                                              | Creates a character string that can be used in place of the file-name argument in other function calls. |

(continued on next page)

**Table 2–1 (Cont.) I/O Functions and Macros**

| Function or Macro                               | Description                                                                        |
|-------------------------------------------------|------------------------------------------------------------------------------------|
| <b>Terminal I/O—Reading from Files</b>          |                                                                                    |
| <code>getchar, getwchar</code>                  | Reads a single character from the standard input ( <code>stdin</code> ).           |
| <code>gets</code>                               | Reads a line from the standard input ( <code>stdin</code> ).                       |
| <code>scanf, wscanf, vscanf, vwscanf</code>     | Performs formatted input from the standard input.                                  |
| <b>Terminal I/O—Writing to Files</b>            |                                                                                    |
| <code>printf, wprintf, vprintf, vwprintf</code> | Performs formatted output to the standard output ( <code>stdout</code> ).          |
| <code>putchar, putwchar</code>                  | Writes a single character to the standard output and returns the character.        |
| <code>puts</code>                               | Writes a character string to the standard output followed by a new-line character. |

## 2.1 Using RMS from RTL Routines

When you create a file using the *Compaq C* RTL I/O functions and macros, you can supply values for many RMS file attributes, including:

- Allocation quantity
- Block size
- Default file extension
- Default file name
- File access context options
- File-processing options
- File-sharing options
- Multiblock count
- Multibuffer count
- Maximum record size
- Record attributes
- Record format
- Record-processing options

See the description of the `creat` function in the Reference Section for information on these values.

Other functions that allow you to set these values include `open`, `fopen`, and `freopen`.

For more information about RMS, see the *Compaq C User's Guide for OpenVMS Systems*.



## 2.2 UNIX I/O and Standard I/O

UNIX I/O functions are UNIX system services, now standardized by ISO POSIX-1 (the ISO Portable Operating System Interface).

UNIX I/O functions use file descriptors to access files. A *file descriptor* is an integer that identifies the file. A file descriptor is declared in the following way, where *file\_desc* is the name of the file descriptor:

```
int file_desc;
```

UNIX I/O functions, such as `creat`, associate the file descriptor with a file. Consider the following example:

```
file_desc1 = creat("INFILE.DAT", 0, "rat=cr", "rfm=var");
```

This statement creates the file, `INFILE.DAT`, with file access mode 0, carriage-return control, variable-length records, and it associates the variable `file_desc1` with the file. When the file is accessed for other operations, such as reading or writing, the file descriptor is used to refer to the file. For example:

```
write(file_desc1, buffer, sizeof(buffer));
```

This statement writes the contents of the buffer to `INFILE.DAT`.

There may be circumstances when you should use UNIX I/O functions and macros instead of the Standard I/O functions and macros. For a detailed discussion of both forms of I/O and how they manipulate the RMS file formats, see Chapter 1.

Standard I/O functions are specified by the ANSI C Standard.

Standard I/O functions add buffering to the features of UNIX I/O and use file pointers to access files. A *file pointer* is an object of type `FILE *`, which is a typedef defined in the `<stdio.h>` header file as follows:

```
typedef struct _iobuf *FILE;
```

The `_iobuf` identifier is also defined in `<stdio.h>`.

To declare a file pointer, use the following:

```
FILE *file_ptr;
```

You use the Standard I/O `fopen` function to create or open an existing file. For example:

```
#include <stdio.h>

main()
{
 FILE *outfile;
 outfile = fopen("DISKFILE.DAT", "w+");
 .
 .
 .
}
```

Here, the file `DISKFILE.DAT` is opened for write-update access.

The *Compaq C* RTL provides the following functions for converting between file descriptors and file pointers:

- `fileno`—returns the file descriptor associated with the specified file pointer.
- `fdopen`—associates a file pointer with a file descriptor returned by an `open`, `creat`, `dup`, `dup2`, or `pipe` function.

## 2.3 Wide-Character Versus Byte I/O Functions

The wide-character I/O functions provide operations similar to most of the byte I/O functions, except that the fundamental units internal to the wide-character functions are wide characters.

However, the external representation (in files) is a sequence of multibyte characters, not wide characters. For the wide-character formatted input and output functions:

- The wide-character formatted *input* functions (such as `fwscanf`) always read a sequence of multibyte characters from files, regardless of the specified directive and, before any further processing, convert this sequence to a sequence of wide characters.
- The wide-character formatted *output* functions (such as `fwprintf`) write wide characters to output files by first converting wide-character argument types to a sequence of multibyte characters, then calling the underlying operating system output primitives.

Byte I/O functions cannot handle state-dependent encodings. Wide-character I/O functions can. They accomplish this by associating each wide-character stream with a conversion-state object of type `mbstate_t`.

The wide-character I/O functions are:

|                       |                       |                      |                        |                      |
|-----------------------|-----------------------|----------------------|------------------------|----------------------|
| <code>fgetwc</code>   | <code>fputwc</code>   | <code>fwscanf</code> | <code>fwprintf</code>  | <code>ungetwc</code> |
| <code>fgetws</code>   | <code>fputws</code>   | <code>wscanf</code>  | <code>wprintf</code>   |                      |
| <code>getwc</code>    | <code>putwc</code>    |                      | <code>vfwprintf</code> |                      |
| <code>getwchar</code> | <code>putwchar</code> |                      | <code>vwprintf</code>  |                      |

The byte I/O functions are:

|                      |                      |                     |                       |                     |
|----------------------|----------------------|---------------------|-----------------------|---------------------|
| <code>fgetc</code>   | <code>fputc</code>   | <code>fscanf</code> | <code>fprintf</code>  | <code>ungetc</code> |
| <code>fgets</code>   | <code>fputs</code>   | <code>scanf</code>  | <code>printf</code>   | <code>fread</code>  |
| <code>getc</code>    | <code>putc</code>    |                     | <code>vfprintf</code> | <code>fwrite</code> |
| <code>gets</code>    | <code>puts</code>    |                     | <code>vprintf</code>  |                     |
| <code>getchar</code> | <code>putchar</code> |                     |                       |                     |

The wide-character input functions read multibyte characters from the stream and convert them to wide characters as if they were read by successive calls to the `fgetwc` function. Each conversion occurs as a call were made to the `mbrtowc` function with the conversion state described by the stream's own `mbstate_t` object.

The wide-character output functions convert wide characters to multibyte characters and write them to the stream as if they were written by successive calls to the `fputwc` function. Each conversion occurs as if a call were made to the `wcrtomb` function, with the conversion state described by the I/O stream's own `mbstate_t` object.

If a wide-character I/O function encounters an invalid multibyte character, the function sets `errno` to the value `EILSEQ`.

## 2.4 Conversion Specifications

Several of the Standard I/O functions (including the Terminal I/O functions) use conversion specifications to specify data formats for I/O. These functions are the formatted-input and formatted-output functions. Consider the following example:

```
int x = 5.0;
FILE *outfile;
.
.
.
fprintf(outfile, "The answer is %d.\n", x);
```

The decimal value of the variable `x` replaces the conversion specification `%d` in the string to be written to the file associated with the identifier `outfile`.

Each conversion specification begins with a percent sign (`%`) and ends with a *conversion specifier*, which is a character that specifies the type of conversion to be performed. Optional characters can appear between the percent sign and the conversion specifier.

For the wide-character formatted I/O functions, the conversion specification is a string of wide characters. For the byte I/O equivalent functions, it is a string of bytes.

Sections 2.4.1 and 2.4.2 describe these optional characters and conversion specifiers.

### 2.4.1 Converting Input Information

The format specification string for the input of information can include three kinds of items:

- White-space characters (spaces, tabs, and new-line characters), which match optional white-space characters in the input field.
- Ordinary characters (not `%`), which must match the next nonwhite-space character in the input.
- Conversion specifications, which govern the conversion of the characters in an input field and their assignment to an object indicated by a corresponding input pointer.

Each input pointer is an address expression indicating an object whose type matches that of a corresponding conversion specification. Conversion specifications form part of the format string. The indicated object is the target that receives the input value. There must be as many input pointers as there are conversion specifications, and the addressed objects must match the types of the conversion specifications.

A conversion specification consists of the following characters, in the order listed:

- A percent character (`%`) or the sequence `%n$` (where *n* is an integer),  
The sequence `%n$` denotes that the conversion is applied to the *n*th input pointer listed, where *n* is a decimal integer between `[1, NL_ARGMAX]` (see the `<limits.h>` header file). For example, a conversion specification beginning with `%5$` means that the conversion will be applied to the fifth input pointer listed after the format specification. The sequence `%%` is invalid.

If the conversion specification does not begin with the sequence `%n$`, the conversion specification is matched to its input pointer in left-to-right order. You should only use one type of conversion specification (`%` or `%n$`) in a format specification.

- One or more optional characters (described in Table 2–2).
- A conversion specifier (described in Table 2–3).

Table 2–2 shows the characters you can use between the percent sign (`%`) (or the sequence `%n$`), and the conversion specifier. These characters are optional but, if specified, must occur in the order shown in Table 2–2.

**Table 2–2 Optional Characters Between `%` (or `%n$`) and the Input Conversion Specifier**

| Character                                                                 | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>*</code>                                                            | An assignment-suppressing character.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| field width                                                               | <p>A nonzero decimal integer that specifies the maximum field width.</p> <p>For the wide-character input functions, the field width is measured in wide characters.</p> <p>For the byte input functions, the field width is measured in bytes, unless the directive is one of the following:</p> <p><code>%lc</code>, <code>%ls</code>, <code>%C</code>, <code>%S</code>, <code>%[</code></p> <p>In these cases, the field width is measured in multibyte character units.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>h</code> , <code>l</code> , or <code>L</code> (or <code>ll</code> ) | <p>Precede a conversion specifier of <code>d</code>, <code>i</code>, or <code>n</code> with an <code>h</code> if the corresponding argument is a pointer to <code>short int</code> rather than a pointer to <code>int</code>; with an <code>l</code> (lowercase ell) if it is a pointer to <code>long int</code>; or, for <i>OpenVMS Alpha</i> systems only, with an <code>L</code> or <code>ll</code> (two lowercase ell) if it is a pointer to <code>__int64</code>.</p> <p>Precede a conversion specifier of <code>o</code>, <code>u</code>, or <code>x</code> with an <code>h</code> if the corresponding argument is a pointer to <code>unsigned short int</code> rather than a pointer to <code>unsigned int</code>; with an <code>l</code> if it is a pointer to <code>unsigned long int</code>; or, for <i>OpenVMS Alpha</i> systems only, with an <code>L</code> or <code>ll</code> if it is a pointer to <code>unsigned __int64</code>.</p> <p>Precede a conversion specifier of <code>c</code>, <code>s</code>, or <code>[</code> with an <code>l</code> (lower ell) if the corresponding argument is a pointer to a <code>wchar_t</code>.</p> <p>Finally, precede a conversion specifier of <code>e</code>, <code>f</code>, or <code>g</code> with an <code>l</code> (lowercase ell) if the corresponding argument is a pointer to <code>double</code> rather than a pointer to <code>float</code>, or with an <code>L</code> if it is a pointer to <code>long double</code>.</p> <p>If an <code>h</code>, <code>l</code>, <code>L</code>, or <code>ll</code> appears with any other conversion specifier, then the behavior is undefined.</p> |

Table 2–3 describes the conversion specifiers for formatted input.

**Table 2–3 Conversion Specifiers for Formatted Input**

| Specifier | Input Type <sup>1</sup> | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| d         |                         | Expects a decimal integer in the input whose format is the same as expected for the subject sequence of the <code>strtol</code> function with the value 10 for the <code>base</code> argument. The corresponding argument must be a pointer to <code>int</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| i         |                         | Expects an integer whose type is determined by the leading input characters. A leading 0 is equated to octal, a leading 0X or 0x is equated to hexadecimal, and all other forms are equated to decimal. The corresponding argument must be a pointer to <code>int</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| o         |                         | Expects an octal integer in the input (with or without a leading 0). The corresponding argument must be a pointer to <code>int</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| u         |                         | Expects a decimal integer in the input whose format is the same as expected for the subject sequence of the <code>strtoul</code> function with the value 10 for the <code>base</code> argument.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| x         |                         | Expects a hexadecimal integer in the input (with or without a leading 0x). The corresponding argument must be a pointer to <code>unsigned int</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| c         | Byte                    | <p>Expects a single byte in the input. The corresponding argument must be a pointer to <code>char</code>.</p> <p>If a field width precedes the <code>c</code> conversion specifier, then the number of characters specified by the field width is read. In this case, the corresponding argument must be a pointer to an array of <code>char</code>.</p> <p>If the optional character <code>l</code> (lowercase ell) precedes this conversion specifier, then the specifier expects a multibyte character in the input which is converted into a wide-character code.</p> <p>The corresponding argument must be a pointer to type <code>wchar_t</code>. If a field width also precedes the <code>c</code> conversion specifier, then the number of characters specified by the field width is read. In this case, the corresponding argument must be a pointer to an array of <code>wchar_t</code>.</p> |
|           | Wide-character          | <p>Expects a sequence of the number of characters specified in the optional field width; this is 1 if not specified.</p> <p>If no <code>l</code> (lowercase ell) precedes the <code>c</code> specifier, then the corresponding argument must be a pointer to an array of <code>char</code>.</p> <p>If an <code>l</code> (lowercase ell) precedes the <code>c</code> specifier, then the corresponding argument must be a pointer to an array of <code>wchar_t</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                               |

<sup>1</sup>Either *byte* or *wide-character*. Where neither is shown for a given specifier, the specifier description applies to both.

(continued on next page)

**Table 2–3 (Cont.) Conversion Specifiers for Formatted Input**

| Specifier | Input Type <sup>1</sup> | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| C         | Byte                    | <p>The specifier expects a multibyte character in the input, which is converted into a wide-character code. The corresponding argument must be a pointer to type <code>wchar_t</code>.</p> <p>If a field width also precedes the C conversion specifier, then the number of characters specified by the field width is read. In this case, the corresponding argument must be a pointer to an array of <code>wchar_t</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                        |
|           | Wide-character          | <p>Expects a sequence of the number of characters specified in the optional field width; this is 1 if not specified. The corresponding argument must be a pointer to an array of <code>wchar_t</code>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| s         | Byte                    | <p>Expects a sequences of bytes in the input. The corresponding argument must be a pointer to an array of characters that is large enough to contain the sequence and a terminating null character (<code>\0</code>) that is automatically added. The input field is terminated by a space, tab, or new-line character.</p> <p>If the optional character l (ell) precedes this conversion specifier, then the specifier expects a sequence of multibyte characters in the input, which are converted to wide-character codes. The corresponding argument must be a pointer to an array of wide characters (type <code>wchar_t</code>) that is large enough to contain the sequence plus the terminating null wide-character code that is automatically added. The input field is terminated by a space, tab, or new-line character.</p> |
|           | Wide-character          | <p>Expects (conceptually) a sequence of nonwhite-space characters in the input.</p> <p>If no l (lowercase ell) precedes the s specifier, then the corresponding argument must be a pointer to an array of <code>char</code> large enough to contain the sequence plus the terminating null byte that is automatically added.</p> <p>If an l (lowercase ell) precedes the s specifier, then the corresponding argument must be a pointer to an array of <code>wchar_t</code> large enough to contain the sequence plus the terminating null wide character that is automatically added.</p>                                                                                                                                                                                                                                              |
| S         | Byte                    | <p>The specifier expects a sequence of multibyte characters in the input, which are converted to wide-character codes. The corresponding argument must be a pointer to an array of wide characters (type <code>wchar_t</code>) that is large enough to contain the sequence plus a terminating null wide-character code that is added automatically. The input field is terminated by a space, tab, or new-line character.</p>                                                                                                                                                                                                                                                                                                                                                                                                          |
|           | Wide-character          | <p>Expects a sequence of nonwhite-space characters in the input. The corresponding argument must be a pointer to an array of <code>wchar_t</code> large enough to contain the sequence plus the terminating null wide character that is automatically added.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

<sup>1</sup>Either *byte* or *wide-character*. Where neither is shown for a given specifier, the specifier description applies to both.

(continued on next page)

**Table 2–3 (Cont.) Conversion Specifiers for Formatted Input**

| Specifier | Input Type <sup>1</sup> | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| e, f, g   |                         | <p>Expects a floating-point number in the input. The corresponding argument must be a pointer to <code>float</code>. The input format for floating-point numbers is: <code>[\pm]nnn[radix][ddd][{E   e}[\pm]nn]</code>. The <code>n</code>'s and <code>d</code>'s are decimal digits (as many as indicated by the field width minus the signs and the letter <code>E</code>). The radix character is defined in the current locale.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| [ . . . ] |                         | <p>Expects a nonempty sequence of characters that is not delimited by a white-space character. The brackets enclose a set of characters (the <i>scanset</i>) expected in the input sequence. Any character in the input sequence that does not match a character in the scanset terminates the character sequence.</p> <p>All characters between the brackets comprise the scanset, unless the first character after the left bracket is a circumflex (<code>^</code>). In this case, the scanset contains all characters other than those that appear between the circumflex and the right bracket. Any character that <i>does</i> appear between the circumflex and the right bracket will terminate the input character sequence.</p> <p>If the conversion specifier begins with <code>[]</code> or <code>[^]</code>, then the right bracket character is in the scanset and the next right bracket character is the matching right bracket that ends the specification; otherwise, the first right bracket character ends the specification.</p> |
|           | Byte                    | <p>If an <code>l</code> (lowercase ell) does not precede the <code>[]</code> specifier, then the characters in the scanset must be single-byte characters only. In this case, the corresponding argument must be a pointer to an array of <code>char</code> large enough to accept the sequence and the terminating null byte that is automatically added.</p> <p>If an <code>l</code> (lowercase ell) does precede the <code>[]</code> specifier, then the characters in the input sequence are considered to be multibyte characters, which are then converted to a wide-character sequence for further processing. If character ranges are specified in the scanset, then the processing is done according to the <code>LC_COLLATE</code> category of the current program's locale. In this case, the corresponding argument must be a pointer to an array of <code>wchar_t</code> large enough to accept the sequence and the terminating null wide character that is automatically added.</p>                                                   |

<sup>1</sup>Either *byte* or *wide-character*. Where neither is shown for a given specifier, the specifier description applies to both.

(continued on next page)

**Table 2–3 (Cont.) Conversion Specifiers for Formatted Input**

| Specifier | Input Type <sup>1</sup> | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           | Wide-character          | <p>If no l (lowercase ell) precedes the [ conversion specifier, then processing is the same as described for the Byte-input type of the %[ specifier, except that the corresponding argument must be an array of <code>char</code> large enough to accept the multibyte sequence plus the terminating null byte that is automatically added.</p> <p>If an l (lowercase ell) precedes the [ conversion specifier, then processing is the same as in the preceding paragraph except that the corresponding argument must be an array of <code>wchar_t</code> large enough to accept the wide-character sequence plus the terminating null wide character that is automatically added.</p> |
| p         |                         | Requires an argument that is a pointer to <code>void</code> . The input value is interpreted as a hexadecimal value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| n         |                         | No input is consumed. The corresponding argument is a pointer to an integer. The integer is assigned the number of characters read from the input stream so far by this call to the formatted input function. Execution of a <code>%n</code> directive does not increment the assignment count returned when the formatted input function completes execution.                                                                                                                                                                                                                                                                                                                          |
| %         |                         | Matches a single percent symbol. No conversion or assignment takes place. The complete conversion specification would be <code>%%</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

<sup>1</sup>Either *byte* or *wide-character*. Where neither is shown for a given specifier, the specifier description applies to both.

### Remarks

- You can change the delimiters of the input field with the bracket ([ ]) conversion specification. Otherwise, an input field is defined as a string of nonwhite-space characters. It extends either to the next white-space character or until the field width, if specified, is exhausted. The function reads across line and record boundaries, since the new-line character is a white-space character.
- A call to one of the input conversion functions resumes searching immediately after the last character processed by a previous call.
- If the assignment-suppression character (\*) appears in the format specification, no assignment is made. The corresponding input field is interpreted and then skipped.
- The arguments must be pointers or other address-valued expressions, since *Compaq C* permits only calls by value. To read a number in decimal format and assign its value to `n`, you must use the following form:

```
scanf("%d", &n)
```

You cannot use the following form:

```
scanf("%d", n)
```

- White space in a format specification matches optional white space in the input field. Consider the following format specification:

```
field = %x
```



This format specification matches the following forms:

```
field = 5218
field=5218
field= 5218
field =5218
```

These forms do not match the following example:

```
fiel d=5218
```

## 2.4.2 Converting Output Information

The format specification string for the output of information can contain:

- Ordinary characters, which are copied to the output.
- Conversion specifications, each of which causes the conversion of a corresponding output source to a character string in a particular format. Conversion specifications are matched to output sources in left-to-right order.

A conversion specification consists of the following, in the order listed:

- A percent character (%) or the sequence %n\$.

The sequence %n\$ denotes that the conversion is applied to the *n*th output source listed, where *n* is a decimal integer between [1, NL\_ARGMAX] (see the <limits.h> header file). For example, a conversion specification beginning with %5\$ means that the conversion will be applied to the fifth output source listed after the format specification.

If the conversion specification does not begin with the sequence %n\$, the conversion specification is matched to its output source in left-to-right order. You should only use one type of conversion specification (% or %n\$) in a format specification.

- One or more optional characters (described in Table 2–4).
- A conversion specifier (described in Table 2–5) concludes the conversion specification.

For examples of conversion specifications, see the sample programs in Section 2.6.

Table 2–4 shows the characters you can use between the percent sign (%) (or the sequence %n\$) and the conversion specifier. These characters are optional, but if specified, they must occur in the order shown in Table 2–4.

**Table 2–4 Optional Characters Between % (or %n\$) and the Output Conversion Specifier**

| Character        | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| flags            | You can use the following flag characters, alone or in any combined order, to modify the conversion specification:                                                                                                                                                                                                                                                                                                                                                                                                                |
| ' (single quote) | Requests that a numeric conversion is formatted with the thousands separator character. Only the numbers to the left of the radix character are formatted with the separator character. The character used as a separator and the positioning of the separators are defined in the program's current locale.                                                                                                                                                                                                                      |
| – (hyphen)       | Left-justifies the converted output source in its field.                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| +                | Requests that an explicit sign be present on a signed conversion. If this flag is not specified, the result of a signed conversion begins with a sign only when a negative value is converted.                                                                                                                                                                                                                                                                                                                                    |
| <i>space</i>     | Prefixes a space to the result of a signed conversion, if the first character of the conversion is not a sign, or if the conversion results in no characters. If you specify both the <i>space</i> and the + flag, the <i>space</i> flag is ignored.                                                                                                                                                                                                                                                                              |
| #                | Requests an alternate conversion format. Depending on the conversion specified, different actions will occur.<br>For the o (octal) conversion, the precision is increased to force the first digit to be a zero.<br>For the x (or X) conversion, a nonzero result is prefixed with 0x (or 0X).<br>For e, E, f, g, and G conversions, the result contains a decimal point even at the end of an integer value.<br>For g and G conversions, trailing zeros are not trimmed.<br>For other conversions, the effect of # is undefined. |
| 0                | Uses zeros rather than spaces to pad the field width for d, i, o, u, x, X, e, E, f, g, and G conversions. If both the 0 and the – flags are specified, then the 0 flag is ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag is ignored. For other conversions, the behavior of the 0 flag is undefined.                                                                                                                                                                                      |

(continued on next page)

**Table 2–4 (Cont.) Optional Characters Between % (or %n\$) and the Output Conversion Specifier**

| Character   | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| field width | <p>The minimum field width can be designated by a decimal integer constant, or by an output source. To specify an output source, use an asterisk ( <code>*</code> ) or the sequence <code>*n\$</code>, where <i>n</i> refers to the <i>n</i>th output source listed after the format specification.</p> <p>The minimum field width is considered after the conversion is done according to all the other components of the format directive. This component affects the padding of the conversion result as follows:</p> <p>If the result of the conversion is wider than the minimum field, write it out.</p> <p>If the result of the conversion is narrower than the minimum width, pad it to make up the field width. Pad with spaces by default. Pad with zeros if the <code>0</code> flag is specified; this does not mean that the width is an octal number. Padding is on the left by default, and on the right if a minus sign is specified.</p> <p>For the wide-character output functions, the field width is measured in wide characters; for the byte output functions, it is measured in bytes.</p> |
| . (period)  | Separates the field width from the precision.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| precision   | <p>The precision defines any of the following:</p> <ul style="list-style-type: none"> <li>• Minimum number of digits to appear for <code>d</code>, <code>i</code>, <code>o</code>, <code>u</code>, <code>x</code>, and <code>X</code> conversions</li> <li>• Number of digits to appear after the decimal-point character for <code>e</code>, <code>E</code>, and <code>f</code> conversions</li> <li>• Maximum number of significant digits for <code>g</code> and <code>G</code> conversions</li> <li>• Maximum number of characters to be written from a string in an <code>s</code> or <code>S</code> conversion</li> </ul> <p>If a precision appears with any other conversion specifier, the behavior is undefined.</p> <p>Precision can be designated by a decimal integer constant, or by an output source. To specify an output source, use an asterisk ( <code>*</code> ) or the sequence <code>*n\$</code>, where <i>n</i> refers to the <i>n</i>th output source listed after the format specification.</p> <p>If only the period is specified, the precision is taken as 0.</p>                     |

(continued on next page)

**Table 2–4 (Cont.) Optional Characters Between % (or %n\$) and the Output Conversion Specifier**

| Character          | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| h, l, or L (or ll) | <p>An h specifies that a following d, i, o, u, x, or X conversion specifier applies to a <code>short int</code> or <code>unsigned short int</code> argument; an h can also specify that a following n conversion specifier applies to a pointer to a <code>short int</code> argument.</p> <p>An l (lowercase ell) specifies that a following d, i, o, u, x, or X conversion specifier applies to a <code>long int</code> or <code>unsigned long int</code> argument; an l can also specify that a following n conversion specifier applies to a pointer to a <code>long int</code> argument.</p> <p>On <i>OpenVMS Alpha</i> systems, an L or ll (two lowercase ells) specifies that a following d, i, o, u, x, or X conversion specifier applies to an <code>__int64</code> or <code>unsigned __int64</code> argument. (<i>Alpha only</i>)</p> <p>An L specifies that a following e, E, f, g, or G conversion specifier applies to a <code>long double</code> argument.</p> <p>An l specifies that a following c or s conversion specifier applies to a <code>wchar_t</code> argument.</p> <p>If an h, l, or L appears with any other conversion specifier, the behavior is undefined.</p> <p>On <i>OpenVMS VAX</i> and <i>OpenVMS Alpha</i> systems, <i>Compaq C</i> <code>int</code> values are equivalent to <code>long</code> values.</p> |

Table 2–5 describes the conversion specifiers for formatted output.

**Table 2–5 Conversion Specifiers for Formatted Output**

| Specifier Output Type <sup>1</sup> | Description                                                                                                                                                                                                                             |
|------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| d, i                               | Converts an <code>int</code> argument to signed decimal format.                                                                                                                                                                         |
| o                                  | Converts an <code>unsigned int</code> argument to unsigned octal format.                                                                                                                                                                |
| u                                  | Converts an <code>unsigned int</code> argument to unsigned decimal format (giving a number in the range 0 to 4,294,967,295).                                                                                                            |
| x, X                               | Converts an <code>unsigned int</code> argument to unsigned hexadecimal format (with or without a leading 0x). The letters <code>abcdef</code> are used for x conversion, and the letters <code>ABCDEF</code> are used for X conversion. |

<sup>1</sup>Either *byte* or *wide-character*. Where neither is shown for a given specifier, the specifier description applies to both.

(continued on next page)

**Table 2–5 (Cont.) Conversion Specifiers for Formatted Output**

| Specifier | Output Type <sup>1</sup> | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| f         |                          | <p>Converts a <code>float</code> or <code>double</code> argument to the format <code>[-]mmm.nnnnnn</code>. The number of <code>n</code>'s is equal to the precision specification:</p> <ul style="list-style-type: none"> <li>• If no precision is specified, the default is 6.</li> <li>• If the precision is 0 and the <code>#</code> flag is specified, the decimal point appears but no <code>n</code>'s appear.</li> <li>• If the precision is 0 and the <code>#</code> flag is not specified, the decimal point also does not appear.</li> <li>• If a decimal point appears, at least one digit appears before it.</li> </ul> <p>The value is rounded to the appropriate number of digits.</p>                   |
| e, E      |                          | <p>Converts a <code>float</code> or <code>double</code> argument to the format <code>[-]m.nnnnnnE±xx</code>. The number of <code>n</code>'s is specified by the precision. If no precision is specified, the default is 6. If the precision is explicitly 0 and the <code>#</code> flag is specified, the decimal point appears but no <code>n</code>'s appear. If the precision is explicitly 0 and the <code>#</code> flag is not specified, the decimal point also does not appear. An <code>'e'</code> is printed for <code>e</code> conversion; an <code>'E'</code> is printed for <code>E</code> conversion. The exponent always contains at least two digits. If the value is 0, the exponent is 0.</p>         |
| g, G      |                          | <p>Converts a <code>float</code> or <code>double</code> argument to format <code>f</code> or <code>e</code> (or <code>E</code> if the <code>G</code> conversion specifier is used), with the precision specifying the number of significant digits. If the precision is 0, it is taken as 1. The format used depends on the value of the argument: format <code>e</code> (or <code>E</code>) is used only if the exponent resulting from such a conversion is less than <code>-4</code>, or is greater than or equal to the precision; otherwise, format <code>f</code> is used. Trailing zeros are suppressed in the fractional portion of the result. A decimal point appears only if it is followed by a digit.</p> |
| c         | Byte                     | <p>Converts an <code>int</code> argument to an unsigned <code>char</code>, and writes the resulting byte.</p> <p>If the optional character <code>l</code> (lowercase ell) precedes this conversion specifier, then the specifier converts a <code>wchar_t</code> argument to an array of bytes representing the character, and writes the resulting character. If the field width is specified and the resulting character occupies fewer bytes than the field width, then it will be padded to the given width with space characters. If the precision is specified, then the behavior is undefined.</p>                                                                                                              |

<sup>1</sup>Either *byte* or *wide-character*. Where neither is shown for a given specifier, the specifier description applies to both.

(continued on next page)

**Table 2–5 (Cont.) Conversion Specifiers for Formatted Output**

| Specifier | Output Type <sup>1</sup> | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| C         | Wide-character           | <p>If an l (lowercase ell) does not precede the c specifier, then the <code>int</code> argument is converted to a wide character as if by calling <code>btowc</code>, and the resulting character is written.</p> <p>If an l (lowercase ell) precedes the c specifier, then the specifier converts a <code>wchar_t</code> argument to an array of bytes representing the character, and writes the resulting character. If the field width is specified and the resulting character occupies fewer characters than the field width, it will be padded to the given width with space characters. If the precision is specified, the behavior is undefined.</p>                                                                                                                                                                                                                                                                                                                                                                             |
|           | Byte                     | <p>Converts a <code>wchar_t</code> argument to an array of bytes representing the character, and writes the resulting character. If the field width is specified and the resulting character occupies fewer bytes than the field width, then it will be padded to the given width with space characters. If the precision is specified, then the behavior is undefined.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|           | Wide-character           | <p>Converts a <code>wchar_t</code> argument to an array of bytes representing the character, and writes the resulting character. If the field width is specified and the resulting character occupies fewer wide characters than the field width, then it will be padded to the given width with space characters. If the precision is specified, then the behavior is undefined.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| s         | Byte                     | <p>Requires an argument that is a pointer to an array of characters of type <code>char</code>. The argument is used to write characters until a null character is encountered or until the number of characters indicated by the precision specification is exhausted. If the precision specification is 0 or omitted, then all characters up to a null are output.</p> <p>If the optional character l (lowercase ell) precedes this conversion specifier, then the specifier converts an array of wide-character codes to multibyte characters, and writes the multibyte characters. Requires an argument that is a pointer to an array of wide characters of type <code>wchar_t</code>. Characters are written until a null wide character is encountered or until the number of bytes indicated by the precision specification is exhausted. If the precision specification is omitted or is greater than the size of the array of converted bytes, then the array of wide characters must be terminated by a null wide character.</p> |

<sup>1</sup>Either *byte* or *wide-character*. Where neither is shown for a given specifier, the specifier description applies to both.

(continued on next page)

**Table 2–5 (Cont.) Conversion Specifiers for Formatted Output**

| Specifier      | Output Type <sup>1</sup> | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | Wide-character           | <p>If an <code>l</code> (lowercase ell) does not precede the <code>s</code> specifier, then the specifier converts an array of multibyte characters, as if by calling <code>mbrtowc</code> for each multibyte character, and writes the resulting characters until a null wide character is encountered or the number of wide characters indicated by the precision specification is exhausted. If the precision specification is omitted or is greater than the size of the array of converted characters, then the converted array must be terminated by a null wide character.</p> <p>If an <code>l</code> precedes this conversion specifier, then the argument is a pointer to an array of <code>wchar_t</code>. Characters from this array are written until a null wide character is encountered or the number of wide characters indicated by the precision specification is exhausted. If the precision specification is omitted or is greater than the size of the array, then the array must be terminated by a null wide character.</p> |
| <code>S</code> | Byte                     | Converts an array of wide-character codes to multibyte characters, and writes the multibyte characters. Requires an argument that is a pointer to an array of wide characters of type <code>wchar_t</code> . Characters are written until a null wide character is encountered or until the number of bytes indicated by the precision specification is exhausted. If the precision specification is omitted or is greater than the size of the array of converted bytes, then the array of wide characters must be terminated by a null wide character.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|                | Wide-character           | The argument is a pointer to an array of <code>wchar_t</code> . Characters from this array are written until a null wide character is encountered or the number of wide characters indicated by the precision specification is exhausted. If the precision specification is omitted or is greater than the size of the array, then the array must be terminated by a null wide character.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <code>p</code> |                          | Requires an argument that is a pointer to <code>void</code> . The value of the pointer is output as a hexadecimal number.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <code>n</code> |                          | Requires an argument that is a pointer to an integer. The integer is assigned the number of characters written to the output stream so far by this call to the formatted output function. No argument is converted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <code>%</code> |                          | Writes out the percent symbol. No conversion is performed. The complete conversion specification would be <code>%%</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

<sup>1</sup>Either *byte* or *wide-character*. Where neither is shown for a given specifier, the specifier description applies to both.

## 2.5 Terminal I/O

*Compaq C* defines three file pointers that allow you to perform I/O to and from the logical devices usually associated with your terminal (for interactive jobs) or a batch stream (for batch jobs). In the *OpenVMS* environment, the three permanent process files `SYS$INPUT`, `SYS$OUTPUT`, and `SYS$ERROR` perform the same functions for both interactive and batch jobs. Terminal I/O refers to both terminal and batch stream I/O. The file pointers `stdin`, `stdout`, and `stderr` are defined when you include the `<stdio.h>` header file using the `#include` preprocessor directive.

The `stdin` file pointer is associated with the terminal to perform input. This file is equivalent to `SYS$INPUT`. The `stdout` file pointer is associated with the terminal to perform output. This file is equivalent to `SYS$OUTPUT`. The `stderr` file pointer is associated with the terminal to report run-time errors. This file is equivalent to `SYS$ERROR`.

There are three file descriptors that refer to the terminal. The file descriptor 0 is equivalent to `SYS$INPUT`, 1 is equivalent to `SYS$OUTPUT`, and 2 is equivalent to `SYS$ERROR`.

When performing I/O at the terminal, you can use Standard I/O functions and macros (specifying the pointers `stdin`, `stdout`, or `stderr` as arguments), you can use UNIX I/O functions (giving the corresponding file descriptor as an argument), or you can use the Terminal I/O functions and macros. There is no functional advantage to using one type of I/O over another; the Terminal I/O functions might save keystrokes since there are no arguments.

## 2.6 Program Examples

This section gives some program examples that show how the I/O functions can be used in applications.

Example 2–1 shows the `printf` function.

### Example 2–1 Output of the Conversion Specifications

```
/* CHAP_2_OUT_CONV.C */
/* This program uses the printf function to print the */
/* various conversion specifications and their effect */
/* on the output. */
/* Include the proper header files in case printf has */
/* to return EOF. */

#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>

#define WIDE_STR_SIZE 20

main()
{
 double val = 123345.5;
 char c = 'C';
 int i = -1500000000;
 char *s = "thomasina";
 wchar_t wc;
 wchar_t ws[WIDE_STR_SIZE];

 /* Produce a wide character and a wide character string */
 if (mbtowc(&wc, "W", 1) == -1) {
 perror("mbtowc");
 exit(EXIT_FAILURE);
 }

 if (mbstowcs(ws, "THOMASINA", WIDE_STR_SIZE) == -1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }
}
```

(continued on next page)



### Example 2–1 (Cont.) Output of the Conversion Specifications

```
/* Print the specification code, a colon, two tabs, and the */
/* formatted output value delimited by the angle bracket */
/* characters (<>). */

printf("%9.4f:\t\t<%9.4f>\n", val);
printf("%9f:\t\t<%9f>\n", val);
printf("%9.0f:\t\t<%9.0f>\n", val);
printf("%-9.0f:\t\t<%-9.0f>\n\n", val);

printf("%11.6e:\t\t<%11.6e>\n", val);
printf("%11e:\t\t<%11e>\n", val);
printf("%11.0e:\t\t<%11.0e>\n", val);
printf("%-11.0e:\t\t<%-11.0e>\n\n", val);

printf("%11g:\t\t<%11g>\n", val);
printf("%9g:\t\t<%9g>\n\n", val);

printf("%d:\t\t<%d>\n", c);
printf("%c:\t\t<%c>\n", c);
printf("%o:\t\t<%o>\n", c);
printf("%x:\t\t<%x>\n\n", c);

printf("%d:\t\t<%d>\n", i);
printf("%u:\t\t<%u>\n", i);
printf("%x:\t\t<%x>\n\n", i);

printf("%s:\t\t<%s>\n", s);
printf("%-9.6s:\t\t<%-9.6s>\n", s);
printf("%-*.s:\t\t<%-*.s>\n", 9, 5, s);
printf("%6.0s:\t\t<%6.0s>\n\n", s);
printf("%cC:\t\t<%C>\n", wc);
printf("%sS:\t\t<%S>\n", ws);
printf("%-9.6S:\t\t<%-9.6S>\n", ws);
printf("%-*.sS:\t\t<%-*.sS>\n", 9, 5, ws);
printf("%6.0S:\t\t<%6.0S>\n\n", ws);
}
```

Running Example 2–1 produces the following output:

```
$ RUN EXAMPLE
%9.4f: <123345.5000>
%9f: <123345.500000>
%9.0f: < 123346>
%-9.0f: <123346 >

%11.6e: <1.233455e+05>
%11e: <1.233455e+05>
%11.0e: < 1e+05>
%-11.0e: <1e+05 >

%11g: < 123346>
%9g: < 123346>

%d: <67>
%c: <C>
%o: <103>
%x: <43>

%d: <-1500000000>
%u: <2794967296>
%x: <a697d100>

%s: <thomasina>
%-9.6s: <thomas >
%-*.s: <thoma >
%6.0s: < >
```

```

%C: <W>
%S: <THOMASINA>
%-9.6S: <THOMAS >
%-.*.S: <THOMA >
%6.0S: < >
$

```

Example 2–2 shows the use of the `fopen`, `ftell`, `sprintf`, `fputs`, `fseek`, `fgets`, and `fclose` functions.

### Example 2–2 Using the Standard I/O Functions

```

/* CHAP_2_STDIO.C */

/* This program establishes a file pointer, writes lines from
/* a buffer to the file, moves the file pointer to the second
/* record, copies the record to the buffer, and then prints
/* the buffer to the screen. */

#include <stdio.h>
#include <stdlib.h>

main()
{
 char buffer[32];
 int i,
 pos;
 FILE *fptr;

 /* Set file pointer. */
 fptr = fopen("data.dat", "w+");
 if (fptr == NULL) {
 perror("fopen");
 exit(EXIT_FAILURE);
 }

 for (i = 1; i < 5; i++) {
 if (i == 2) /* Get position of record 2. */
 pos = ftell(fptr);
 /* Print a line to the buffer. */
 sprintf(buffer, "test data line %d\n", i);
 /* Print buffer to the record. */
 fputs(buffer, fptr);
 }

 /* Go to record number 2. */
 if (fseek(fptr, pos, 0) < 0) {
 perror("fseek"); /* Exit on fseek error. */
 exit(EXIT_FAILURE);
 }

 /* Read record 2 in the buffer. */
 if (fgets(buffer, 32, fptr) == NULL) {
 perror("fgets"); /* Exit on fgets error. */
 exit(EXIT_FAILURE);
 }

 /* Print the buffer. */
 printf("Data in record 2 is: %s", buffer);
 fclose(fptr); /* Close the file. */
}

```

Running Example 2–2 produces the following result:

```

$ RUN EXAMPLE
Data in record 2 is: test data line 2

```

The output to DATA.DAT from Example 2-2 is:

```
test data line 1
test data line 2
test data line 3
test data line 4
```

### Example 2-3 Using Wide Character I/O Functions

```
/* CHAP_2_WC_IO.C */
/* This program establishes a file pointer, writes lines from */
/* a buffer to the file using wide-character codes, moves the */
/* file pointer to the second record, copies the record to the */
/* wide-character buffer, and then prints the buffer to the */
/* screen. */

#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>

main()
{
 char flat_buffer[32];
 wchar_t wide_buffer[32];
 wchar_t format[32];
 int i,
 pos;
 FILE *fptr;

 /* Set file pointer. */
 fptr = fopen("data.dat", "w+");
 if (fptr == NULL) {
 perror("fopen");
 exit(EXIT_FAILURE);
 }

 for (i = 1; i < 5; i++) {
 if (i == 2) /* Get position of record 2. */
 pos = ftell(fptr);
 /* Print a line to the buffer. */
 sprintf(flat_buffer, "test data line %d\n", i);
 if (mbstowcs(wide_buffer, flat_buffer, 32) == -1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 /* Print buffer to the record. */
 fputws(wide_buffer, fptr);
 }

 /* Go to record number 2. */
 if (fseek(fptr, pos, 0) < 0) {
 perror("fseek"); /* Exit on fseek error. */
 exit(EXIT_FAILURE);
 }
}
```

(continued on next page)

### Example 2–3 (Cont.) Using Wide Character I/O Functions

```
/* Put record 2 in the buffer. */
if (fgetws(wide_buffer, 32, fptr) == NULL) {
 perror("fgetws"); /* Exit on fgets error. */
 exit(EXIT_FAILURE);
}
/* Print the buffer. */
printf("Data in record 2 is: %S", wide_buffer);
fclose(fptr); /* Close the file. */
}
```

Running Example 2–3 produces the following result:

```
$ RUN EXAMPLE
Data in record 2 is: test data line 2
```

The output to DATA.DAT from Example 2–3 is:

```
test data line 1
test data line 2
test data line 3
test data line 4
```

Example 2–4 shows the use of both a file pointer and a file descriptor to access a single file.

### Example 2–4 I/O Using File Descriptors and Pointers

```
/* CHAP_2_FILE_DIS_AND_POINTER.C */
/* The following example creates a file with variable-length */
/* records (rfm=var) and the carriage-return attribute (rat=cr). */
/* */
/* The program uses creat to create and open the file, and */
/* fdopen to associate the file descriptor with a file pointer. */
/* After using the fdopen function, the file must be referenced */
/* using the Standard I/O functions. */

#include <unixio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define ERROR 0
#define ERROR1 -1
#define BUFFSIZE 132

main()
{
 char buffer[BUFFSIZE];
 int fildes;
 FILE *fp;

 if ((fildes = creat("data.dat", 0, "rat=cr",
 "rfm=var")) == ERROR1) {
 perror("DATA.DAT: creat() failed\n");
 exit(EXIT_FAILURE);
 }
}
```

(continued on next page)

#### Example 2–4 (Cont.) I/O Using File Descriptors and Pointers

```
if ((fp = fdopen(fildes, "w")) == NULL) {
 perror("DATA.DAT: fdopen() failed\n");
 exit(EXIT_FAILURE);
}
while (fgets(buffer, BUFSIZE, stdin) != NULL)
 if (fwrite(buffer, strlen(buffer), 1, fp) == ERROR) {
 perror("DATA.DAT: fwrite() failed\n");
 exit(EXIT_FAILURE);
 }
if (fclose(fp) == EOF) {
 perror("DATA.DAT: fclose() failed\n");
 exit(EXIT_FAILURE);
}
}
```



## Character, String, and Argument-List Functions

Table 3–1 describes the character, string, and argument-list functions in the *Compaq C* Run-Time Library (RTL). Although further discussion follows in this chapter, see the Reference Section for more detailed information on each function.

**Table 3–1 Character, String, and Argument-List Functions**

| Function                         | Description                                                                                            |
|----------------------------------|--------------------------------------------------------------------------------------------------------|
| <b>Character Classification</b>  |                                                                                                        |
| <code>isalnum, iswalnum</code>   | Returns a nonzero integer if its argument is one of the alphanumeric characters in the current locale. |
| <code>isalpha, iswalpha</code>   | Returns a nonzero integer if its argument is one of the alphabetic characters in the current locale.   |
| <code>isascii</code>             | Returns a nonzero integer if its argument is any ASCII character.                                      |
| <code>isctrl, iswctrl</code>     | Returns a nonzero integer if its argument is a control character in the current locale.                |
| <code>isdigit, iswdigit</code>   | Returns a nonzero integer if its argument is a digit character in the current locale.                  |
| <code>isgraph, iswgraph</code>   | Returns a nonzero integer if its argument is a graphic character in the current locale.                |
| <code>islower, iswlower</code>   | Returns a nonzero integer if its argument is a lowercase character in the current locale.              |
| <code>isprint, iswprint</code>   | Returns a nonzero integer if its argument is a printing character in the current locale.               |
| <code>ispunct, iswpunct</code>   | Returns a nonzero integer if its argument is a punctuation character in the current locale.            |
| <code>isspace, iswspace</code>   | Returns a nonzero integer if its argument is a white-space character in the current locale.            |
| <code>isupper, iswupper</code>   | Returns a nonzero integer if its argument is an uppercase character in the current locale.             |
| <code>isxdigit, iswxdigit</code> | Returns a nonzero integer if its argument is a hexadecimal digit (0 to 9, A to F, or a to f).          |

(continued on next page)

**Table 3–1 (Cont.) Character, String, and Argument-List Functions**

| Function                                      | Description                                                                                                                             |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <b>Character Conversion</b>                   |                                                                                                                                         |
| <code>btowc</code>                            | Converts a one-byte multibyte character to a wide character in the initial shift state.                                                 |
| <code>ecvt, fcvt, gcvt</code>                 | Converts an argument to a null-terminated string of ASCII digits and return the address of the string.                                  |
| <code>index, rindex</code>                    | Searches for a character in a string.                                                                                                   |
| <code>mblen, mbrlen</code>                    | Determines the number of bytes in a multibyte character.                                                                                |
| <code>mbsinit</code>                          | Determines whether an <code>mbstate_t</code> object describes an initial conversion state.                                              |
| <code>mbstowcs</code>                         | Converts a sequence of multibyte characters into a sequence of corresponding codes.                                                     |
| <code>toascii</code>                          | Converts its argument, an 8-bit ASCII character, to a 7-bit ASCII character.                                                            |
| <code>tolower, _tolower, towlower</code>      | Converts its argument, an uppercase character, to lowercase.                                                                            |
| <code>toupper, _toupper, towupper</code>      | Converts its argument, a lowercase character, to uppercase.                                                                             |
| <code>towctrans</code>                        | Maps one wide character to another according to a specified mapping descriptor.                                                         |
| <code>wctob</code>                            | Determines if a wide character corresponds to a single-byte multibyte character and returns its multibyte character representation.     |
| <code>wcstombs</code>                         | Converts a sequence of wide-character codes corresponding to multibyte characters to a sequence of multibyte characters.                |
| <code>wctomb</code>                           | Converts a wide character to its multibyte character representation.                                                                    |
| <code>wctrans</code>                          | Returns the description of a mapping, corresponding to specified property, that can be later used in a call to <code>towctrans</code> . |
| <b>String Manipulation</b>                    |                                                                                                                                         |
| <code>atof</code>                             | Converts a given string to a double-precision number.                                                                                   |
| <code>atoi, atol</code>                       | Converts a given string of ASCII characters to the appropriate numeric values.                                                          |
| <code>atoll, atoll (Alpha only)</code>        | Converts a given string of ASCII characters to an <code>__int64</code> .                                                                |
| <code>basename</code>                         | Returns the last component of a path name.                                                                                              |
| <code>dirname</code>                          | Reports the parent directory name of a file path name.                                                                                  |
| <code>strcat, strncat, wscat, wcsncat</code>  | Appends one string to the end of another string.                                                                                        |
| <code>strchr, strrchr, wcschr, wcsrchr</code> | Returns the address of the first or last occurrence of a given character in a null-terminated string.                                   |

(continued on next page)



**Table 3–1 (Cont.) Character, String, and Argument-List Functions**

| Function                                                        | Description                                                                                                                                                                                                                           |
|-----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>String Manipulation</b>                                      |                                                                                                                                                                                                                                       |
| <code>strcmp, strncmp, strcoll, wcscmp, wcsncmp, wcscoll</code> | Compares two character strings and return a negative, zero, or positive integer indicating that the values of the individual characters in the first string are less than, equal to, or greater than the values in the second string. |
| <code>strcpy, strncpy, wcsncpy</code>                           | Copies all or part of one string into another.                                                                                                                                                                                        |
| <code>strxfrm, wcsxfrm</code>                                   | Transforms a multibyte string to another string ready for comparisons using the <code>strcmp</code> or <code>wcscmp</code> function.                                                                                                  |
| <code>strcspn, wcsbspn</code>                                   | Searches a string for a character that is in a specified set of characters.                                                                                                                                                           |
| <code>strlen, wcslen</code>                                     | Returns the length of a string of characters. The returned length does not include the terminating null character ( <code>\0</code> ).                                                                                                |
| <code>strpbrk, wcpbrk</code>                                    | Searches a string for the occurrence of one of a specified set of characters.                                                                                                                                                         |
| <code>strspn, wcsspn</code>                                     | Searches a string for the occurrence of a character that is not in a specified set of characters.                                                                                                                                     |
| <code>strstr, wcs wcs</code>                                    | Searches a string for the first occurrence of a specified set of characters.                                                                                                                                                          |
| <code>strtod, wcstod</code>                                     | Converts a given string to a double-precision number.                                                                                                                                                                                 |
| <code>strtok, wcstok</code>                                     | Locates text tokens in a given string.                                                                                                                                                                                                |
| <code>strtol, wcstol</code>                                     | Converts the initial portion of a string to a signed long integer.                                                                                                                                                                    |
| <code>strtoll, strtouq</code><br>(Alpha only)                   | Converts the initial portion of a string to signed <code>__int64</code> .                                                                                                                                                             |
| <code>strtoul, wcstoul</code>                                   | Converts the initial portion of a string to an unsigned long integer.                                                                                                                                                                 |
| <code>strtoull, strtouq</code><br>(Alpha only)                  | Converts the initial portion of the string pointed to by the pointer to the character string to an unsigned <code>__int64</code> .                                                                                                    |
| <b>String Handling—Accessing Binary Data</b>                    |                                                                                                                                                                                                                                       |
| <code>bcmp</code>                                               | Compares byte strings.                                                                                                                                                                                                                |
| <code>bcopy</code>                                              | Copies byte strings.                                                                                                                                                                                                                  |
| <code>bzero</code>                                              | Copies nulls into byte strings.                                                                                                                                                                                                       |
| <code>memchr, wmemchr</code>                                    | Locates the first occurrence of the specified byte within the initial length of the object to be searched.                                                                                                                            |
| <code>memcmp, wmemcmp</code>                                    | Compares two objects byte by byte.                                                                                                                                                                                                    |
| <code>memcpy, memmove, wmemcpy, wmemmove</code>                 | Copies a specified number of bytes from one object to another.                                                                                                                                                                        |
| <code>memset, wmemset</code>                                    | Sets a specified number of bytes in a given object to a given value.                                                                                                                                                                  |

(continued on next page)

**Table 3–1 (Cont.) Character, String, and Argument-List Functions**

| Function                                                                | Description                                                                                                |
|-------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| <b>Argument-List Handling—Accessing a Variable-Length Argument List</b> |                                                                                                            |
| va_arg                                                                  | Returns the next item in the argument list.                                                                |
| va_count                                                                | Returns the number of longwords ( <i>VAX only</i> ) or quadwords ( <i>VAX only</i> ) in the argument list. |
| va_end                                                                  | Finishes the va_start session.                                                                             |
| va_start,<br>va_start_1                                                 | Initializes a variable to the beginning of the argument list.                                              |
| vfprintf, vprintf,<br>vsprintf                                          | Prints formatted output based on an argument list.                                                         |

**3.1 Character-Classification Functions**

The character-classification functions take a single argument on which they perform a logical operation. The argument can have any value; it does not have to be an ASCII character. The isascii function determines if the argument is an ASCII character (0 through 177 octal). The other functions determine whether the argument is a particular type of ASCII character, such as a graphic character or digit. The isw\* functions test wide characters. Character-classification information is in the LC\_CTYPE category of the program’s current locale.

For all functions, a positive return value indicates TRUE. A return value of 0 indicates FALSE.

To briefly reference the character-classification functions in a subsequent table, each function is assigned a number, as shown in Table 3–2.

**Table 3–2 Character-Classification Functions**

| Function Number | Function | Function Number | Function |
|-----------------|----------|-----------------|----------|
| 1               | isalnum  | 7               | islower  |
| 2               | isalpha  | 8               | isprint  |
| 3               | isascii  | 9               | ispunct  |
| 4               | iscntrl  | 10              | isspace  |
| 5               | isdigit  | 11              | isupper  |
| 6               | isgraph  | 12              | isxdigit |

Table 3–3 lists the numbers of the functions (as assigned in Table 3–2) that return the value TRUE for each of the given ASCII characters. The numeric code represents the octal value of each of the ASCII characters.

**Table 3–3 ASCII Characters and the Character-Classification Functions**

| <b>ASCII Values</b> | <b>Function Numbers</b> | <b>ASCII Values</b> | <b>Function Numbers</b> |
|---------------------|-------------------------|---------------------|-------------------------|
| NUL 00              | 3,4                     | @ 100               | 3,6,8,9                 |
| SOH 01              | 3,4                     | A 101               | 1,2,3,6,8,11,12         |
| STX 02              | 3,4                     | B 102               | 1,2,3,6,8,11,12         |
| ETX 03              | 3,4                     | C 103               | 1,2,3,6,8,11,12         |
| EOT 04              | 3,4                     | D 104               | 1,2,3,6,8,11,12         |
| ENQ 05              | 3,4                     | E 105               | 1,2,3,6,8,11,12         |
| ACK 06              | 3,4                     | F 106               | 1,2,3,6,8,11,12         |
| BEL 07              | 3,4                     | G 107               | 1,2,3,6,8,11            |
| BS 10               | 3,4                     | H 110               | 1,2,3,6,8,11            |
| HT 11               | 3,4,10                  | I 111               | 1,2,3,6,8,11            |
| LF 12               | 3,4,10                  | J 112               | 1,2,3,6,8,11            |
| VT 13               | 3,4,10                  | K 113               | 1,2,3,6,8,11            |
| FF 14               | 3,4,10                  | L 114               | 1,2,3,6,8,11            |
| CR 15               | 3,4,10                  | M 115               | 1,2,3,6,8,11            |
| SO 16               | 3,4                     | N 116               | 1,2,3,6,8,11            |
| SI 17               | 3,4                     | O 117               | 1,2,3,6,8,11            |
|                     |                         |                     |                         |
| DLE 20              | 3,4                     | P 120               | 1,2,3,6,8,11            |
| DC1 21              | 3,4                     | Q 121               | 1,2,3,6,8,11            |
| DC2 22              | 3,4                     | R 122               | 1,2,3,6,8,11            |
| DC3 23              | 3,4                     | S 123               | 1,2,3,6,8,11            |
| DC4 24              | 3,4                     | T 124               | 1,2,3,6,8,11            |
| NAK 25              | 3,4                     | U 125               | 1,2,3,6,8,11            |
| SYN 26              | 3,4                     | V 126               | 1,2,3,6,8,11            |
| ETB 27              | 3,4                     | W 127               | 1,2,3,6,8,11            |
|                     |                         |                     |                         |
| CAN 30              | 3,4                     | X 130               | 1,2,3,6,8,11            |
| EM 31               | 3,4                     | Y 131               | 1,2,3,6,8,11            |
| SUB 32              | 3,4                     | Z 132               | 1,2,3,6,8,11            |
| ESC 33              | 3,4                     | [ 133               | 3,6,8,9                 |
| FS 34               | 3,4                     | \ 134               | 3,6,8,9                 |
| GS 35               | 3,4                     | ] 135               | 3,6,8,9                 |
| RS 36               | 3,4                     | ^ 136               | 3,6,8,9                 |
| US 37               | 3,4                     | – 137               | 3,6,8,9                 |
|                     |                         |                     |                         |
| SP 40               | 3,8,10                  | ? 140               | 3,6,8,9                 |
| ! 41                | 3,6,8,9                 | a 141               | 1,2,3,6,7,8,12          |

(continued on next page)

**Table 3–3 (Cont.) ASCII Characters and the Character-Classification Functions**

| <b>ASCII Values</b> | <b>Function Numbers</b> | <b>ASCII Values</b> | <b>Function Numbers</b> |
|---------------------|-------------------------|---------------------|-------------------------|
| " 42                | 3,6,8,9                 | b 142               | 1,2,3,6,7,8,12          |
| # 43                | 3,6,8,9                 | c 143               | 1,2,3,6,7,8,12          |
| \$ 44               | 3,6,8,9                 | d 144               | 1,2,3,6,7,8,12          |
| % 45                | 3,6,8,9                 | e 145               | 1,2,3,6,7,8,12          |
| & 46                | 3,6,8,9                 | f 146               | 1,2,3,6,7,8,12          |
| ' 47                | 3,6,8,9                 | g 147               | 1,2,3,6,7,8             |
| ( 50                | 3,6,8,9                 | h 150               | 1,2,3,6,7,8             |
| ) 51                | 3,6,8,9                 | i 151               | 1,2,3,6,7,8             |
| * 52                | 3,6,8,9                 | j 152               | 1,2,3,6,7,8             |
| + 53                | 3,6,8,9                 | k 153               | 1,2,3,6,7,8             |
| ' 54                | 3,6,8,9                 | l 154               | 1,2,3,6,7,8             |
| - 55                | 3,6,8,9                 | m 155               | 1,2,3,6,7,8             |
| ? 56                | 3,6,8,9                 | n 156               | 1,2,3,6,7,8             |
| / 57                | 3,6,8,9                 | o 157               | 1,2,3,6,7,8             |
| 0 60                | 1,3,5,6,8,12            | p 160               | 1,2,3,6,7,8             |
| 1 61                | 1,3,5,6,8,12            | q 161               | 1,2,3,6,7,8             |
| 2 62                | 1,3,5,6,8,12            | r 162               | 1,2,3,6,7,8             |
| 3 63                | 1,3,5,6,8,12            | s 163               | 1,2,3,6,7,8             |
| 4 64                | 1,3,5,6,8,12            | t 164               | 1,2,3,6,7,8             |
| 5 65                | 1,3,5,6,8,12            | u 165               | 1,2,3,6,7,8             |
| 6 66                | 1,3,5,6,8,12            | v 166               | 1,2,3,6,7,8             |
| 7 67                | 1,3,5,6,8,12            | w 167               | 1,2,3,6,7,8             |
| 8 70                | 1,3,5,6,8,12            | x 170               | 1,2,3,5,6,8             |
| 9 71                | 1,3,5,6,8,12            | y 171               | 1,2,3,5,6,8             |
| : 72                | 3,6,8,9                 | z 172               | 1,2,3,5,6,8             |
| ; 73                | 3,6,8,9                 | { 173               | 3,6,8,9                 |
| < 74                | 3,6,8,9                 | 174                 | 3,6,8,9                 |
| = 75                | 3,6,8,9                 | } 175               | 3,6,8,9                 |
| > 76                | 3,6,8,9                 | ?~ 176              | 3,6,8,9                 |
| ? 77                | 3,6,8,9                 | DEL 177             | 3,4                     |

Example 3–1 shows how the character-classification functions are used.

### Example 3–1 Character-Classification Functions

```
/* CHAP_3_CHARCLASS.C */
/* This example uses the isalpha, isdigit, and isspace */
/* functions to count the number of occurrences of letters, */
/* digits, and white-space characters entered through the */
/* standard input (stdin). */

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
 char c;
 int i = 0,
 j = 0,
 k = 0;

 while ((c = getchar()) != EOF) {
 if (isalpha(c))
 i++;
 if (isdigit(c))
 j++;
 if (isspace(c))
 k++;
 }

 printf("Number of letters: %d\n", i);
 printf("Number of digits: %d\n", j);
 printf("Number of spaces: %d\n", k);
}
```

The sample input and output from Example 3–1 follows:

```
$ RUN EXAMPLE1
I saw 35 people riding bicycles on Main Street.

Number of letters: 36
Number of digits: 2
Number of spaces: 8
$
```

## 3.2 Character-Conversion Functions

The character-conversion functions convert one type of character to another type. These functions include:

|           |           |
|-----------|-----------|
| ecvt      | tolower   |
| fcvt      | toupper   |
| gcvt      | toupper   |
| mbtowc    | towctrans |
| mbrtowc   | wctrans   |
| mbsrtowcs | wcrtomb   |
| toascii   | wcsrtombs |
| tolower   |           |

For more information on each of these functions, see the Reference Section.

Example 3–2 shows how to use the `ecvt` function.

### Example 3–2 Converting Double Values to an ASCII String

```
/* CHAP_3_CHARCONV.C */
/* This program uses the ecvt function to convert a double */
/* value to a string. The program then prints the string. */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

main()
{
 double val; /* Value to be converted */
 int sign, /* Variables for sign */
 point; /* and decimal place */

 /* Array for the converted string */
 static char string[20];
 val = -3.1297830e-10;
 printf("original value: %e\n", val);
 if (sign)
 printf("value is negative\n");
 else
 printf("value is positive\n");
 printf("decimal point at %d\n", point);
}
```

The output from Example 3–2 is as follows:

```
$ RUN EXAMPLE2
original value: -3.129783e-10
converted string: 31298
value is negative
decimal point at -9
$
```

Example 3–3 shows how to use the `toupper` and `tolower` functions.

### Example 3–3 Changing Characters to and from Uppercase Letters

```
/* CHAP_3_CONV_UPPERLOWER.C */
/* This program uses the functions toupper and tolower to */
/* convert uppercase to lowercase and lowercase to uppercase */
/* using input from the standard input (stdin). */
#include <ctype.h>
#include <stdio.h> /* To use EOF identifier */
#include <stdlib.h>

main()
{
 char c,
 ch;
```

(continued on next page)

### Example 3–3 (Cont.) Changing Characters to and from Uppercase Letters

```
while ((c = getchar()) != EOF) {
 if (c >= 'A' && c <= 'Z')
 ch = tolower(c);
 else
 ch = toupper(c);
 putchar(ch);
}
```

Sample input and output from Example 3–3 are as follows:

```
$ RUN EXAMPLE3
LET'S GO TO THE welcome INN. Ctrl/Z
let's go to the WELCOME inn.
$
```

## 3.3 String and Argument-List Functions

The *Compaq C* RTL contains a group of functions that manipulate strings. Some of these functions concatenate strings; others search a string for specific characters or perform some other comparison, such as determining the equality of two strings.

The *Compaq C* RTL also contains a set of functions that allow you to copy buffers containing binary data.

The set of functions defined and declared in the `<varargs.h>` and the `<stdarg.h>` header files provide a method of accessing variable-length argument lists. The `<stdarg.h>` functions are defined by the ANSI C Standard and are more portable than those defined in `<varargs.h>`.

The RTL functions such as `printf` and `execl`, for example, use variable-length argument lists. User-defined functions with variable-length argument lists that do not use `<varargs.h>` or `<stdarg.h>` are not portable due to the different argument-passing conventions of various machines.

The `<stdarg.h>` header file does not contain `va_alist` and `va_dcl`. The following shows a syntax example when using `<stdarg.h>`:

```
function_name(int arg1, ...)
{
```

```
 va_list ap;
 ...
```

When using `<varargs.h>`:

- The identifier `va_alist` is a parameter in the function definition.
- `va_dcl` declares the parameter `va_alist`, a declaration that is not terminated with a semicolon (;).
- The type `va_list` is used in the declaration of the variable used to traverse the list. You must declare at least one variable of type `va_list` when using `<varargs.h>`.

These names and declarations have the following syntax:

```
function_name(int arg1, ...)
{
 va_list ap;
 .
 .
 .
}
```

## 3.4 Program Examples

Example 3–4 shows how to use the `strcat` and `strncat` functions.

### Example 3–4 Concatenating Two Strings

```
/* CHAP_3_CONCAT.C */
/* This example uses strcat and strncat to concatenate two */
/* strings. */

#include <stdio.h>
#include <string.h>

main()
{
 static char string1[80] = "Concatenates ";
 static char string2[] = "two strings ";
 static char string3[] = "up to a maximum of characters.";
 static char string4[] = "imum number of characters";

 printf("strcat:\t%s\n", strcat(string1, string2));
 printf("strncat (0):\t%s\n", strncat(string1, string3, 0));
 printf("strncat (11):\t%s\n", strncat(string1, string3, 11));
 printf("strncat (40):\t%s\n", strncat(string1, string4, 40));
}
```

Example 3–4 produces the following output:

```
$ RUN EXAMPLE1
strcat: Concatenates two strings
strncat (0): Concatenates two strings
strncat (11): Concatenates two strings up to a max
strncat (40): Concatenates two strings up to a maximum number of characters.
$
```

Example 3–5 shows how to use the `strcspn` function.

### Example 3–5 Four Arguments to the `strcspn` Function

```
/* CHAP_3_STRCSPN.C */
/* This example shows how strcspn interprets four */
/* different kinds of arguments. */

#include <stdio.h>

main()
{
 printf("strcspn with null charset: %d\n",
 strcspn("abcdef", ""));
}
```

(continued on next page)



### Example 3–5 (Cont.) Four Arguments to the strcspn Function

```
printf("strcspn with null string: %d\n",
 strcspn("", "abcdef"));

printf("strcspn(\"xabc\", \"abc\"): %d\n",
 strcspn("xabc", "abc"));

printf("strcspn(\"abc\", \"def\"): %d\n",
 strcspn("abc", "def"));
}
```

The sample output, to the file strcspn.out, in Example 3–5 is as follows:

**\$ RUN EXAMPLE2**

```
strcspn with null charset: 6
strcspn with null string: 0
strcspn("xabc","abc"): 1
strcspn("abc","def"): 3
```

Example 3–6 shows how to use the <stdarg.h> functions and definitions.

### Example 3–6 Using the <stdarg.h> Functions and Definitions

```
/* CHAP_3_STDARG.C */
/* This routine accepts a variable number of string arguments, */
/* preceded by a count of the number of such strings. It */
/* allocates enough space in which to concatenate all of the */
/* strings, concatenates them together, and returns the address */
/* of the new string. It returns NULL if there are no string */
/* arguments, or if they are all null strings. */
#include <stdarg.h> /* Include appropriate header files */
#include <stdlib.h> /* for the "example" call in main. */
#include <string.h>
#include <stdio.h>

/* NSTRINGS is the maximum number of string arguments accepted */
/* (arbitrary). */
#define NSTRINGS 10

char *concatenate(int n,...)
{
 va_list ap; /* Declare the argument pointer. */
 char *list[NSTRINGS],
 *string;
 int index = 0,
 size = 0;

 /* Check that the number of arguments is within range. */
 if (n <= 0)
 return NULL;
 if (n > NSTRINGS)
 n = NSTRINGS;

 va_start(ap, n); /* Initialize the argument pointer. */
 do {
 /* Extract the next argument and save it. */
 list[index] = va_arg(ap, char *);
```

(continued on next page)

### Example 3–6 (Cont.) Using the <stdarg.h> Functions and Definitions

```
 size += strlen(list[index]);
 } while (++index < n);
 va_end(ap); /* Terminate use of ap. */
 if (size == 0)
 return NULL;

 string = malloc(size + 1);
 string[0] = '\0';

 /* Append each argument to the end of the growing result */
 /* string. */
 for (index = 0; index < n; ++index)
 strcat(string, list[index]);

 return string;
}

/* An example of calling this routine is */
main() {
 char *ret_string ;
 ret_string = concatenate(7, "This ", "message ", "is ",
 "built with ", "a", " variable arg",
 " list.") ;

 puts(ret_string) ;
}
```

The call to Example 3–6 produces the following output:

This message is built with a variable arg list.

## Error and Signal Handling

Table 4–1 lists and describes all the error- and signal-handling functions found in the *Compaq C* Run-Time Library (RTL). For more detailed information on each function, see the Reference Section.

**Table 4–1 Error- and Signal-Handling Functions**

| Function                 | Description                                                                                                                       |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <code>abort</code>       | Raises the signal SIGABRT that terminates the execution of the program.                                                           |
| <code>assert</code>      | Puts diagnostics into programs.                                                                                                   |
| <code>atexit</code>      | Registers a function to be called at program termination.                                                                         |
| <code>exit, _exit</code> | Terminates the current program.                                                                                                   |
| <code>perror</code>      | Writes a short error message to <code>stderr</code> describing the current <code>errno</code> value.                              |
| <code>strerror</code>    | Maps the error code in <code>errno</code> to an error message string.                                                             |
| <code>alarm</code>       | Sends the signal SIGALARM to the invoking process after the number of seconds indicated by its argument has elapsed.              |
| <code>gsignal</code>     | Generates a specified software signal.                                                                                            |
| <code>kill</code>        | Sends a SIGKILL signal to the process specified by a process ID.                                                                  |
| <code>longjmp</code>     | Transfers control from a nested series of function invocations back to a predefined point without returning normally.             |
| <code>pause</code>       | Causes the process to wait until it receives a signal.                                                                            |
| <code>raise</code>       | Generates a specified signal.                                                                                                     |
| <code>setjmp</code>      | Establishes the context for a later transfer of control from a nested series of function invocations, without returning normally. |
| <code>sigaction</code>   | Specifies the action to take upon delivery of a signal.                                                                           |
| <code>sigaddset</code>   | Adds the specified individual signal.                                                                                             |
| <code>sigblock</code>    | Causes the signals in its argument to be added to the current set of signals being blocked from delivery.                         |
| <code>sigdelset</code>   | Deletes a specified individual signal.                                                                                            |
| <code>sigemptyset</code> | Initializes the signal set to exclude all signals.                                                                                |
| <code>sigfillset</code>  | Initializes the signal set to include all signals.                                                                                |
| <code>sigismember</code> | Tests whether a specified signal is a member of the signal set.                                                                   |
| <code>siglongjmp</code>  | Nonlocal goto with signal handling.                                                                                               |

(continued on next page)

**Table 4–1 (Cont.) Error- and Signal-Handling Functions**

| Function        | Description                                                                                                           |
|-----------------|-----------------------------------------------------------------------------------------------------------------------|
| sigmask         | Constructs the mask for a given signal number.                                                                        |
| signal          | Catches or ignores a signal.                                                                                          |
| sigpause        | Blocks a specified set of signals and then waits for a signal that was not blocked.                                   |
| sigpending      | Examines pending signals.                                                                                             |
| sigprocmask     | Sets the current signal mask.                                                                                         |
| sigsetjmp       | Sets the jump point for a nonlocal goto.                                                                              |
| sigsetmask      | Establishes the signals that are blocked from delivery.                                                               |
| sigstack        | Defines an alternate stack on which to process signals.                                                               |
| sigsuspend      | Atomically changes the set of blocked signals and waits for a signal.                                                 |
| sigvec          | Permanently assigns a handler for a specific signal.                                                                  |
| ssignal         | Allows you to specify the action to be taken when a particular signal is raised.                                      |
| VAXC\$ESTABLISH | Establishes an application exception handler in a way that is compatible with <i>Compaq C</i> RTL exception handling. |

## 4.1 Error Handling

When an error occurs during a call to any of the *Compaq C* RTL functions, the function returns an unsuccessful status. Many RTL routines also set the external variable `errno` to a value that indicates the reason for the failure. You should always check the return value for an error situation.

The `<errno.h>` header file declares `errno` and symbolically defines the possible error codes. By including the `<errno.h>` header file in your program, you can check for specific error codes after a *Compaq C* RTL function call.

At program startup, the value of `errno` is 0. The value of `errno` can be set to a nonzero value by many *Compaq C* RTL functions. It is not reset to 0 by any *Compaq C* RTL function, so it is only valid to use `errno` after a *Compaq C* RTL function call has been made and a failure status returned. Table 4–2 lists the symbolic values that may be assigned to `errno` by the *Compaq C* RTL.

**Table 4–2 The Error Code Symbolic Values**

| Symbolic Constant | Description                    |
|-------------------|--------------------------------|
| E2BIG             | Argument list too long         |
| EACCES            | Permission denied              |
| EADDRINUSE        | Address already in use         |
| EADDRNOTAVAIL     | Can't assign requested address |
| EAFNOSUPPORT      | Address family not supported   |
| EAGAIN            | No more processes              |
| EALIGN            | Alignment error                |

(continued on next page)

**Table 4–2 (Cont.) The Error Code Symbolic Values**

| Symbolic Constant | Description                           |
|-------------------|---------------------------------------|
| EALREADY          | Operation already in progress         |
| EBADF             | Bad file number                       |
| EBADCAT           | Bad message catalogue format          |
| EBADMSG           | Corrupted message detected            |
| EBUSY             | Mount device busy                     |
| ECANCELED         | Operation canceled                    |
| ECHILD            | No children                           |
| ECONNABORTED      | Software caused connection abort      |
| ECONNREFUSED      | Connection refused                    |
| ECONNRESET        | Connection reset by peer              |
| EDEADLK           | Resource deadlock avoided             |
| EDESTADDRREQ      | Destination address required          |
| EDOM              | Math argument                         |
| EDQUOT            | Disk quota exceeded                   |
| EEXIST            | File exists                           |
| EFAIL             | Cannot start operation                |
| EFAULT            | Bad address                           |
| EFBIG             | File too large                        |
| EFTYPE            | Inappropriate operation for file type |
| EHOSTDOWN         | Host is down                          |
| EHOSTUNREACH      | No route to host                      |
| EIDRM             | Identifier removed                    |
| EILSEQ            | Illegal byte sequence                 |
| EINPROGRESS       | Operation now in progress             |
| EINPROG           | Asynchronous operation in progress    |
| EINTR             | Interrupted system call               |
| EINVAL            | Invalid argument                      |
| EIO               | I/O error                             |
| EISCONN           | Socket is already connected           |
| EISDIR            | Is a directory                        |
| ELOOP             | Too many levels of symbolic links     |
| EMFILE            | Too many open files                   |
| EMLINK            | Too many links                        |
| EMSGSIZE          | Message too long                      |
| ENAMETOOLONG      | File name too long                    |
| ENETDOWN          | Network is down                       |
| ENETRESET         | Network dropped connection on reset   |
| ENETUNREACH       | Network is unreachable                |

(continued on next page)

**Table 4–2 (Cont.) The Error Code Symbolic Values**

| Symbolic Constant | Description                       |
|-------------------|-----------------------------------|
| ENFILE            | File table overflow               |
| ENOBUFFS          | No buffer space available         |
| ENODEV            | No such device                    |
| ENOENT            | No such file or directory         |
| ENOEXEC           | Exec format error                 |
| ENOLCK            | No locks available                |
| ENOMEM            | Not enough core                   |
| ENOMSG            | No message of desired type        |
| ENOPROTOPT        | Protocol not available            |
| ENOSPC            | No space left on device           |
| ENOSYS            | Function not implemented          |
| ENOTBLK           | Block device required             |
| ENOTCONN          | Socket is not connected           |
| ENOTDIR           | Not a directory                   |
| ENOTEMPTY         | Directory not empty               |
| ENOTSOCK          | Socket operation on nonsocket     |
| ENOTSUP           | Function not implemented          |
| ENOTTY            | Not a typewriter                  |
| ENWAIT            | No waiting processes              |
| ENXIO             | No such device or address         |
| EOPNOTSUPP        | Operation not supported on socket |
| EPERM             | Not owner                         |
| EPFNOSUPPORT      | Protocol family not supported     |
| EPIPE             | Broken pipe                       |
| EPROCLIM          | Too many processes                |
| EPROTONOSUPPORT   | Protocol not supported            |
| EPROTOPTYPE       | Protocol wrong type for socket    |
| ERANGE            | Result too large                  |
| EREMOTE           | Too many levels of remote in path |
| EROFS             | Read-only file system             |
| ESHUTDOWN         | Can't send after socket shutdown  |
| ESOCKTNOSUPPORT   | Socket type not supported         |
| ESPIPE            | Illegal seek                      |
| ESRCH             | No such process                   |
| ESTALE            | Stale NFS file handle             |
| ETIMEDOUT         | Connection timed out              |
| ETOOMANYREFS      | Too many references: can't splice |
| ETXTBSY           | Text file busy                    |

(continued on next page)

**Table 4–2 (Cont.) The Error Code Symbolic Values**

| Symbolic Constant | Description                                        |
|-------------------|----------------------------------------------------|
| EUSERS            | Too many users                                     |
| EVMSEERR          | <i>OpenVMS</i> specific nontranslatable error code |
| EWOULDBLOCK       | I/O operation would block channel                  |
| EXDEV             | Cross-device link                                  |

You can translate the error codes to a message, similar to that found in UNIX systems, by using the `perror` or `strerror` function. If `errno` is set to `EVMSEERR`, `perror` cannot translate the error code and prints the following message, followed by the *OpenVMS* error message associated with the value:

```
%s:nontranslatable vms error code: xxxxxx vms message:
```

In the message, `%s` is the string you supply to `perror`; `xxxxxx` is the *OpenVMS* condition value.

If `errno` is set to `EVMSEERR`, then the *OpenVMS* condition value is available in the `vaxc$errno` variable declared in the `<errno.h>` header file. The `vaxc$errno` variable is guaranteed to have a valid value only if `errno` is set to `EVMSEERR`; if `errno` is set to a value other than `EVMSEERR`, the value of `vaxc$errno` is undefined.

See the `strerror` function in the Reference Section for another way to translate error codes.

## 4.2 Signal Handling

A signal is a form of software interrupt to the normal execution of a user process. Signals occur as a result of a variety of events, including any of the following:

- Typing Ctrl/C at a terminal
- Certain programming errors
- A call to the `gsignal` or `raise` function
- A wake-up action

### 4.2.1 *OpenVMS* Versus UNIX Terminology

Both *OpenVMS* and UNIX systems provide signal-handling mechanisms that behave differently but use similar terminology. With the *Compaq C* RTL, you can program using either signal-handling mechanism. Before describing the signal-handling routines, some terminology must be established.

The UNIX term for a software interrupt is *signal*. A routine called by the UNIX system to process a signal is termed a *signal handler*.

A software interrupt on an *OpenVMS* system is referred to as a *signal*, *condition*, or *exception*. A routine called by the *OpenVMS* system to process software interrupts is termed a *signal handler*, *condition handler*, or *exception handler*.

To prevent confusion, the terms *signal* and *signal handler* in this manual refer to UNIX interrupts and interrupt processing routines, while the terms *exception* and *exception handler* refer to *OpenVMS* interrupts and interrupt processing routines.

## 4.2.2 UNIX Signals and the *Compaq C* RTL

Signals are represented by mnemonics defined in the `<signal.h>` header file. Table 4–3 lists the supported signal mnemonics and the corresponding event that causes each signal to be generated on the *OpenVMS* operating system.

**Table 4–3 *Compaq C* RTL Signals**

| Name                    | Description              | Generated by                                                    |
|-------------------------|--------------------------|-----------------------------------------------------------------|
| SIGABRT <sup>1</sup>    | Abort                    | abort()                                                         |
| SIGALRM                 | Alarm clock              | Timer AST, alarm routine                                        |
| SIGBUS                  | Bus error                | Access violation or change mode user                            |
| SIGCHLD                 | Child process stopped    | Child process terminated or stopped                             |
| SIGEMT                  | EMT instruction          | Compatibility mode trap or opcode reserved to customer          |
| SIGFPE                  | Floating-point exception | Floating-point overflow/underflow                               |
| SIGHUP                  | Hang up                  | Data set hang up                                                |
| SIGILL <sup>1</sup>     | Illegal instruction      | Illegal instruction, reserved operand, or reserved address mode |
| SIGINT <sup>4</sup>     | Interrupt                | <i>OpenVMS</i> Ctrl/C interrupt                                 |
| SIGIOT <sup>1</sup>     | IOT instruction          | Opcode reserved to customer                                     |
| SIGKILL <sup>2, 3</sup> | Kill                     | External signal only                                            |
| SIGQUIT <sup>5</sup>    | Quit                     | Not implemented.                                                |
| SIGPIPE                 | Broken pipe              | Write to a pipe with no readers.                                |
| SIGSEGV                 | Segment violation        | Length violation or change mode user                            |
| SIGSYS                  | System call error        | Bad argument to system call                                     |
| SIGTERM                 | Software terminate       | External signal only                                            |
| SIGTRAP <sup>1</sup>    | Trace trap               | TBIT trace trap or breakpoint fault instruction                 |
| SIGUSR1                 | User-defined signal      | Explicit program call to raise the signal                       |
| SIGUSR2                 | User-defined signal      | Explicit program call to raise the signal                       |
| SIGWINCH <sup>6</sup>   | Window size changed      | Explicit program call to raise the signal                       |

<sup>1</sup>Cannot be reset when caught.

<sup>2</sup>Cannot be caught or ignored.

<sup>3</sup>Cannot be blocked.

<sup>4</sup>Setting SIGINT can affect processing of Ctrl/Y interrupts. For example, in response to a caller's request to block or ignore SIGINT, the *Compaq C* RTL disables the Ctrl/Y interrupt.

<sup>5</sup>"Not implemented" for SIGQUIT means that there is no external event, including a Ctrl/Y interrupt, that would trigger a SIGQUIT signal, thereby causing a signal handler established for SIGQUIT to be invoked. This signal can be generated only through an appropriate *Compaq C* RTL function, such as `raise`.

<sup>6</sup>Supported on OpenVMS Version 7.3 and higher.

By default, when a signal (except for SIGCHLD) occurs, the process is terminated. However, you can choose to have the signal ignored by using one of the following functions:

`sigaction`



```
signal
sigvec
ssignal
```

You can have the signal blocked by using one of the following functions:

```
sigblock
sigsetmask
sigprocmask
sigsuspend
sigpause
```

Table 4–3 indicates those signals that cannot be ignored or blocked.

You can also establish a signal handler to catch and process a signal by using one of the following functions:

```
sigaction
signal
sigvec
ssignal
```

Unless noted in Table 4–3, each signal can be reset. A signal is reset if the signal handler function calls `signal` or `ssignal` to re-establish itself to catch the signal. Example 4–1 shows how to establish a signal handler and reset the signal.

The calling interface to a signal handler is:

```
void handler (int sigint);
```

Where *sigint* is the signal number of the raised signal that caused this handler to be called.

A signal handler installed with `sigvec` remains installed until it is changed.

A signal handler installed with `signal` or `signal` remains installed until the signal is generated.

A signal handler can be installed for more than one signal. Use the `sigaction` routine with the flag `SA_RESETHAND` to control this.

### 4.2.3 Signal-Handling Concepts

A signal is said to be *generated* for (or sent to) a process when the event that causes the signal first occurs. Examples of such events include detection of hardware faults, timer expiration, and terminal activity, as well as the invocation of `kill`. In some circumstances, the same event generates signals for multiple processes.

Each process has an action to be taken in response to each signal defined by the system. A signal is said to be *delivered* to a process when the appropriate action for the process and signal is taken.

During the time between the generation of a signal and its delivery, the signal is said to be *pending*. Ordinarily, this interval cannot be detected by an application. However, a signal can be *blocked* from delivery to a process:

- If the action associated with a blocked signal is anything other than to ignore the signal, and if that signal is generated for the process, the signal remains pending until either it is unblocked or the action associated with it is set to ignore the signal.

- If the action associated with a blocked signal is to ignore the signal and if that signal is generated for the process, it is unspecified whether the signal is discarded immediately upon generation or remains pending.

Each process has a *signal mask* that defines the set of signals currently blocked from delivery to it. The signal mask for a process is initialized from that of its parent. The `sigaction`, `sigprocmask`, and `sigsuspend` functions control the manipulation of the signal mask.

The determination of which action is taken in response to a signal is made at the time the signal is delivered, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated. If a subsequent occurrence of a pending signal is generated, it is implementation-dependent as to whether the signal is delivered more than once. The *Compaq C* RTL delivers the signal only once. The order in which multiple, simultaneously pending signals are delivered to a process is unspecified.

#### 4.2.4 Signal Actions

This section applies to the `sigaction`, `signal`, `sigvec`, and `ssignal` functions.

There are three types of action that can be associated with a signal:

```
SIG_DFL
SIG_IGN
pointer to a function
```

Initially, all signals are set to `SIG_DFL` or `SIG_IGN` prior to entry of the main routine (see the `exec` functions.) The actions prescribed by these values are:

`SIG_DFL` — signal-specific default action

- The default actions for the signals defined in this document are specified under `<signal.h>`.
- If the default action is to stop the process, the execution of that process is temporarily suspended. When a process stops, a `SIGCHLD` signal is generated for its parent process, unless the parent process has set the `SA_NOCLDSTOP` flag. While a process is stopped, any additional signals that are sent to the process are not delivered until the process is continued, except `SIGKILL` which always terminates the receiving process. A process that is a member of an orphaned process group is not allowed to stop in response to the `SIGSTOP`, `SIGTTIN`, or `SIGTTOU` signals. In cases where delivery of one of these signals would stop such a process, the signal is discarded.
- Setting a signal action to `SIG_DFL` for a signal that is pending and whose default action is to ignore the signal (for example, `SIGCHLD`), causes the pending signal to be discarded, whether or not it is blocked.

`SIG_IGN` — ignore signal

- Delivery of the signal has no effect on the process. The behavior of a process is undefined after it ignores a `SIGFPE`, `SIGILL`, or `SIGSEGV` signal that was not generated by `kill` or `raise`.
- The system does not allow the action for the `SIGKILL` or `SIGSTOP` signals to be set to `SIG_IGN`.

- Setting a signal action to SIG\_IGN for a signal that is pending causes the pending signal to be discarded, whether or not it is blocked.
- If a process sets the action for the SIGCHLD signal to SIG\_IGN, the behavior is unspecified.

*pointer to a function* — catch signal

- On delivery of the signal, the receiving process executes the signal-catching function at the specified address. After returning from the signal-catching function, the receiving process resumes execution at the point at which it was interrupted.

- Specify the signal-catching function as:

```
void func(int signo);
```

Where *func* is the specified signal-catching function and *signo* is the signal number of the signal being delivered.

- The behavior of a process is undefined after it returns normally from a signal-catching function for a SIGFPE, SIGKILL, or SIGSEGV signal that was not generated by kill or raise.
- The system does not allow a process to catch the signals SIGKILL and SIGSTOP.
- If a process establishes a signal-catching function for the SIGCHLD signal while it has a terminated child process for which it has not waited, it is unspecified whether a SIGCHLD signal is generated to indicate that child process.

#### 4.2.5 Signal Handling and *OpenVMS* Exception Handling

This section discusses how *Compaq C* RTL signal handling is implemented with and interacts with *OpenVMS* exception handling. Information in this section allows you to write *OpenVMS* exception handlers that do not conflict with *Compaq C* RTL signal handling. For information on *OpenVMS* exception handling, see the *OpenVMS Procedure Calling and Condition Handling Standard*.

The *Compaq C* RTL implements signals with *OpenVMS* exceptions. When `gsignal` or `raise` is called, the signal number is translated to a particular *OpenVMS* exception, which is used in a call to `LIB$SIGNAL`. This mechanism is necessary to catch an *OpenVMS* exception resulting from a user error and translate it into a corresponding UNIX signal. For example, an ACCVIO resulting from a write to a NULL pointer is translated to a SIGBUS or SIGSEGV signal.

Tables 4–4 and 4–5 list the *Compaq C* RTL signal names, the corresponding *OpenVMS* VAX and *OpenVMS Alpha* exceptions, the event that generates the signal, and the optional signal code for use with the `gsignal` and `raise` functions.

**Table 4–4   *Compaq C* RTL Signals and Corresponding *OpenVMS* VAX Exceptions** (*VAX only*)

| Name                  | <i>OpenVMS</i> Exception  | Generated By                    | Code             |
|-----------------------|---------------------------|---------------------------------|------------------|
| SIGABRT               | SS\$_OPCCUS               | The <code>abort</code> function | –                |
| SIGALRM               | SS\$_ASTFLT               | The <code>alarm</code> function | –                |
| SIGBUS                | SS\$_ACCVIO               | Access violation                | –                |
| SIGBUS                | SS\$_CMODUSER             | Change mode user                | –                |
| SIGCHLD               | C\$_SIGCHLD               | Child process stopped           | –                |
| SIGEMT                | SS\$_COMPAT               | Compatibility mode trap         | –                |
| SIGFPE                | SS\$_DECOVF               | Decimal overflow trap           | FPE_DECOVF_TRAP  |
| SIGFPE                | SS\$_FLTDIV               | Floating/decimal division by 0  | FPE_FLTDIV_TRAP  |
| SIGFPE                | SS\$_FLTDIV_F             | Floating divide by 0 fault      | FPE_FLTDIV_FAULT |
| SIGFPE                | SS\$_FLTTOVF              | Floating overflow trap          | FPE_FLTOVF_TRAP  |
| SIGFPE                | SS\$_FLTTOVF_F            | Floating overflow fault         | FPE_FLTOVF_FAULT |
| SIGFPE                | SS\$_FLTUND               | Floating undeflow trap          | FPE_FLTUND_TRAP  |
| SIGFPE                | SS\$_FLTUND_F             | Floating undeflow fault         | FPE_FLTUND_FAULT |
| SIGFPE                | SS\$_INTDIV               | Integer division by 0           | FPE_INTDIV_TRAP  |
| SIGFPE                | SS\$_INTOVF               | Integer overflow                | FPE_INTOVF_TRAP  |
| SIGFPE                | SS\$_SUBRNG               | Subscript-range                 | FPE_SUBRNG_TRAP  |
| SIGHUP                | SS\$_HANGUP               | Data set hangup                 | –                |
| SIGILL                | SS\$_OPCDEC               | Reserved instruction            | ILL_PRIVIN_FAULT |
| SIGILL                | SS\$_RADRMOD              | Reserved addressing             | ILL_RESAD_FAULT  |
| SIGILL                | SS\$_ROPRAND              | Reserved operand                | ILL_RESOP_FAULT  |
| SIGINT                | SS\$_CONTROL              | <i>OpenVMS</i> Ctrl/C interrupt | –                |
| SIGIOT                | SS\$_OPCCUS               | Customer-reserved opcode        | –                |
| SIGKILL               | SS\$_ABORT                | External signal only            | –                |
| SIGQUIT               | SS\$_CONTROLY             | The <code>raise</code> function | –                |
| SIGPIPE               | SS\$_NOMBX                | No mailbox                      | –                |
| SIGSEGV               | SS\$_ACCVIO               | Length violation                | –                |
| SIGSEGV               | SS\$_CMODUSER             | Change mode user                | –                |
| SIGSYS                | SS\$_BADPARAM             | Bad argument to system call     | –                |
| SIGTERM               | Not implemented           | –                               | –                |
| SIGTRAP               | SS\$_TBIT                 | TBIT trace trap                 | –                |
| SIGTRAP               | SS\$_BREAK                | Breakpoint fault instruction    | –                |
| SIGUSR1               | C\$_SIGUSR1               | The <code>raise</code> function | –                |
| SIGUSR2               | C\$_SIGUSR2               | The <code>raise</code> function | –                |
| SIGWINCH <sup>1</sup> | C\$_SIGWINCH <sup>2</sup> | The <code>raise</code> function | –                |

<sup>1</sup>Supported on *OpenVMS* Version 7.3 and higher.

<sup>2</sup>SS\$\_BADWINCNT when C\$\_SIGWINCH not defined (*OpenVMS* versions before 7.3).

To call a signal handler that you have established with `signal` or `sigvec`, the *Compaq C* RTL intercepts the *OpenVMS* exceptions that correspond to signals by having an *OpenVMS* exception handler in the main routine of the program. If your program has a main function, then this exception handler is automatically established. If you do not have a main function, or if your main function is written

in a language other than *Compaq C*, then you must invoke the `VAXC$CTRL_INIT` routine to establish this handler.

The *Compaq C* RTL uses *OpenVMS* exceptions to implement the `setjmp` and `longjmp` functions. When the `longjmp` function is called, a `C$_LONGJMP` *OpenVMS* exception is signaled. To prevent the `C$_LONGJMP` exception from being interfered with by user exception handlers, use the `VAXC$ESTABLISH` routine to establish user *OpenVMS* exception handlers instead of calling `LIB$ESTABLISH`. The `C$_LONGJMP` mnemonic is defined in the `<errnode.h>` header file.

If you want to use *OpenVMS* exception handlers and UNIX signals in your C program, your *OpenVMS* exception handler must be prepared to accept and resignal the *OpenVMS* exceptions listed in Tables 4–4 (*VAX only*) and 4–5 (*Alpha only*), as well as the `C$_LONGJMP` exception and any C\$ facility exception that might be introduced in future versions of the *Compaq C* RTL. This is because UNIX signals are global in context, whereas *OpenVMS* exceptions are stack-frame based.

Consequently, an *OpenVMS* exception handler always receives the exception that corresponds to the UNIX signal before the *Compaq C* RTL exception handler in the main routine does. By resignaling the *OpenVMS* exception, you allow the *Compaq C* RTL exception handler to receive the exception. You can intercept any of those *OpenVMS* exceptions yourself, but in doing so you will disable the corresponding UNIX signal.

**Table 4–5** *Compaq C* RTL Signals and Corresponding *OpenVMS Alpha* Exceptions (*Alpha only*)

| Name    | <i>OpenVMS</i> Exception | Generated By                   | Code             |
|---------|--------------------------|--------------------------------|------------------|
| SIGABRT | SS\$_OPCCUS              | The abort function             | –                |
| SIGALRM | SS\$_ASTFLT              | The alarm function             | –                |
| SIGBUS  | SS\$_ACCVIO              | Access violation               | –                |
| SIGBUS  | SS\$_CMODUSER            | Change mode user               | –                |
| SIGCHLD | C\$_SIGCHLD              | Child process stopped          | –                |
| SIGEMT  | SS\$_COMPAT              | Compatibility mode trap        | –                |
| SIGFP   | SS\$_DECDIV              | Decimal divide trap            | FPE_DECDIV_TRAP  |
| SIGFPE  | SS\$_DECINV              | Decimal invalid operand trap   | FPE_DECINV_TRAP  |
| SIGFPE  | SS\$_DECOVF              | Decimal overflow trap          | FPE_DECOVF_TRAP  |
| SIGFPE  | SS\$_HPARITH             | Floating/decimal division by 0 | FPE_FLTDIV_TRAP  |
| SIGFPE  | SS\$_HPARITH             | Floating overflow trap         | FPE_FLTOVF_TRAP  |
| SIGFPE  | SS\$_HPARITH             | Floating undeflow trap         | FPE_FLTUND_TRAP  |
| SIGFPE  | SS\$_HPARITH             | Integer overflow               | FPE_INTOVF_TRAP  |
| SIGFPE  | SS\$_HPARITH             | Invalid operand                | FPE_INVOPR_TRAP  |
| SIGFPE  | SS\$_HPARITH             | Inexact result                 | FPE_INXRES_TRAP  |
| SIGFPE  | SS\$_INTDIV              | Integer div by zero            | FPE_INTDIV_TRAP  |
| SIGFPE  | SS\$_SUBRNG              | Subscript out of range         | FPE_SUBRNG_TRAP  |
| SIGFPE  | SS\$_SUBRNG1             | Subscript1 out of range        | FPE_SUBRNG1_TRAP |

(continued on next page)

**Table 4–5 (Cont.) *Compaq C* RTL Signals and Corresponding *OpenVMS Alpha* Exceptions**

(*Alpha only*)

| Name                  | <i>OpenVMS</i> Exception  | Generated By                    | Code             |
|-----------------------|---------------------------|---------------------------------|------------------|
| SIGFPE                | SS\$_SUBRNG2              | Subscript2 out of range         | FPE_SUBRNG2_TRAP |
| SIGFPE                | SS\$_SUBRNG3              | Subscript3 out of range         | FPE_SUBRNG3_TRAP |
| SIGFPE                | SS\$_SUBRNG4              | Subscript4 out of range         | FPE_SUBRNG4_TRAP |
| SIGFPE                | SS\$_SUBRNG5              | Subscript5 out of range         | FPE_SUBRNG5_TRAP |
| SIGFPE                | SS\$_SUBRNG6              | Subscript6 out of range         | FPE_SUBRNG6_TRAP |
| SIGFPE                | SS\$_SUBRNG7              | Subscript7 out of range         | FPE_SUBRNG7_TRAP |
| SIGHUP                | SS\$_HANGUP               | Data set hangup                 | –                |
| SIGILL                | SS\$_OPCDEC               | Reserved instruction            | ILL_PRIVIN_FAULT |
| SIGILL                | SS\$_ROPRAND              | Reserved operand                | ILL_RESOP_FAULT  |
| SIGINT                | SS\$_CONTROL              | <i>OpenVMS</i> Ctrl/C interrupt | –                |
| SIGIOT                | SS\$_OPCCUS               | Customer-reserved opcode        | –                |
| SIGKILL               | SS\$_ABORT                | External signal only            | –                |
| SIGQUIT               | SS\$_CONTROLY             | The <code>raise</code> function | –                |
| SIGPIPE               | SS\$_NOMBX                | No mailbox                      | –                |
| SIGSEGV               | SS\$_ACCVIO               | Length violation                | –                |
| SIGSEGV               | SS\$_CMODUSER             | Change mode user                | –                |
| SIGSYS                | SS\$_BADPARAM             | Bad argument to system call     | –                |
| SIGTERM               | Not implemented           | –                               | –                |
| SIGTRAP               | SS\$_BREAK                | Breakpoint fault instruction    | –                |
| SIGUSR1               | C\$_SIGUSR1               | The <code>raise</code> function | –                |
| SIGUSR2               | C\$_SIGUSR2               | The <code>raise</code> function | –                |
| SIGWINCH <sup>1</sup> | C\$_SIGWINCH <sup>2</sup> | The <code>raise</code> function | –                |

<sup>1</sup>Supported on *OpenVMS* Version 7.3 and higher.

<sup>2</sup>SS\$\_BADWINCNT when C\$\_SIGWINCH not defined (*OpenVMS* versions before 7.3).

### ***OpenVMS Alpha* Signal-Handling Notes** (*Alpha only*)

- While all signals that exist on *OpenVMS VAX* systems also exist on *OpenVMS Alpha* systems, the corresponding *OpenVMS* exceptions and code is different in a number of cases because on Alpha processors there are two new *OpenVMS* exceptions and several others that are obsolete.
- All floating-point exceptions on *OpenVMS Alpha* systems are signaled by the *OpenVMS* exception SS\$\_HPARITH (high-performance arithmetic trap). The particular type of trap that occurred is translated by the *Compaq C* RTL through use of the exception summary longword, which is set when a high-performance arithmetic trap is signaled.

- Since the `SS$_COMPAT`, `SS$_TBIT`, and `SS$_RADMOD` exceptions are never reported on *OpenVMS Alpha* systems, they are not recognized by the *Compaq C* RTL on *OpenVMS Alpha* systems. Since each signal that corresponds to one of these exceptions also corresponds to another *OpenVMS* exception as well, all the signals are shown in Table 4–5.
- 

## 4.3 Program Example

Example 4–1 shows how the signal, alarm, and pause functions operate. It also shows how to establish a signal handler to catch a signal, which prevents program termination.

### Example 4–1 Suspending and Resuming Programs

```
/* CHAP_4_SUSPEND_RESUME.C */
/* This program shows how to alternately suspend and resume a */
/* program using the signal, alarm, and pause functions. */
#define SECONDS 5
#include <stdio.h>
#include <signal.h>
int number_of_alarms = 5; /* Set alarm counter. */
void alarm_action(int);
main()
{
 signal(SIGALRM, alarm_action); /* Establish a signal handler. */
 /* to catch the SIGALRM signal.*/

 alarm(SECONDS); /* Set alarm clock for 5 seconds. */
 pause(); /* Suspend the process until *
 * the signal is received. */
}
void alarm_action(int x)
{
 printf("\t<%d\007>", number_of_alarms); /* Print the value of */
 /* the alarm counter. */

 signal(SIGALRM, alarm_action); /* Reset the signal. */
 alarm(SECONDS); /* Set the alarm clock. */
 if (--number_of_alarms) /* Decrement alarm counter. */
 pause();
}
```

Here is the sample output from Example 4–1:

```
$ RUN EXAMPLE
<5> <4> <3> <2> <1>
```





---

## Subprocess Functions

The *Compaq C* Run-Time Library (RTL) provides functions that allow you to create subprocesses from a *Compaq C* program. The creating process is called the *parent* and the created subprocess is called the *child*.

To create a child process within the parent process, use the `exec` functions (`execl`, `execle`, `execv`, `execve`, `execlp`, and `execvp`) and the `vfork` function. Other functions are available to allow the parent and child to read and write data across processes (`pipe`) and to allow for synchronization of the two processes (`wait`). This chapter describes how to implement and use these functions.

The parent process can execute *Compaq C* programs in its children, either synchronously or asynchronously. The number of children that can run simultaneously is determined by the `/PRCLM` user authorization quota established for each user on your system. Other quotas that may affect the use of subprocesses are `/ENQLM` (Queue Entry Limit), `/ASTLM` (AST Waits Limit), and `/FILLM` (Open File Limit).

This chapter discusses the subprocess functions. Table 5–1 lists and describes all the subprocess functions found in the *Compaq C* RTL. For more detailed information on each function, see the Reference Section.

**Table 5–1 Subprocess Functions**

| Function                                                                                                                         | Description                                                                          |
|----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <b>Implementation of Child Processes</b>                                                                                         |                                                                                      |
| <code>system</code>                                                                                                              | Passes a given string to the host environment to be executed by a command processor. |
| <code>vfork</code>                                                                                                               | Creates an independent child process.                                                |
| <b>The <code>exec</code> Functions</b>                                                                                           |                                                                                      |
| <code>execl</code> , <code>execle</code> , <code>execlp</code><br><code>execv</code> , <code>execve</code> , <code>execvp</code> | Passes the name of the image to be activated in a child process.                     |
| <b>Synchronizing Process</b>                                                                                                     |                                                                                      |
| <code>wait</code> , <code>wait3</code> , <code>wait4</code> ,<br><code>waitpid</code> ,                                          | Suspends the parent process until a value is returned from a child.                  |
| <b>Interprocess Communication</b>                                                                                                |                                                                                      |
| <code>pipe</code>                                                                                                                | Allows for communication between a parent and child.                                 |

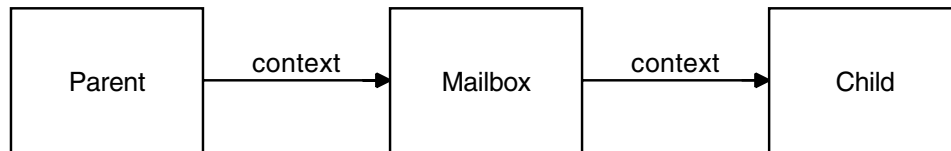
## 5.1 Implementing Child Processes in *Compaq C*

Child processes are created by *Compaq C* functions with the *OpenVMS* LIB\$SPAWN RTL routine. (See the *VMS Run-Time Library Routines Volume* for information on LIB\$SPAWN.) Using LIB\$SPAWN allows you to create multiple levels of child processes; the parent's children can also spawn children, and so on, up to the limits allowed by the user authorization quotas discussed in the introduction to this chapter.

Child processes can only execute other *Compaq C* programs. Other native-mode *OpenVMS* languages do not share the ability of *Compaq C* to communicate between processes; if they do, they do not use the same mechanisms. The parent process must be run under a Compaq supported command-language interpreter (CLI), such as the DIGITAL Command Language (DCL). You cannot run the parent as a detached process or under control of a user-supplied CLI.

Parent and child processes communicate through a mailbox as shown in Figure 5–1. This mailbox transfers the context in which the child will run. This *context mailbox* passes information to the child that it inherits from the parent, such as the names and file descriptors of all the files opened by the parent and the current location within those files. The mailbox is deleted by the parent when the child image exits.

**Figure 5–1 Communications Links Between Parent and Child Processes**



ZK-4002-GE

---

**Note**

The mailbox created by the `vfork` and `exec` functions is temporary. The logical name of this mailbox is VAXC\$EXECMBX and is reserved for use by the *Compaq C* RTL.

---

The mailbox is created with a maximum message size and a buffer quota of 512 bytes each. You need the TMPMBX privilege to create a mailbox with these RTL functions. Since TMPMBX is the privilege required by the DCL commands PRINT and SUBMIT, most users on a system have this privilege. To see what system privileges you have, enter a SHOW PROCESS/PRIVILEGES command.

You cannot change the characteristics of these mailboxes. For more information on mailboxes, see the *VMS I/O User's Reference Volume*.

## 5.2 The exec Functions

There are six exec functions that you can call to execute a *Compaq C* image in the child process. These functions expect that `vfork` has been called to set up a return address. The exec functions will call `vfork` if the parent process did not.

When `vfork` is called by the parent, the exec function returns to the parent process. When `vfork` was called by an exec function, the exec function returns to itself, waits for the child to exit, and then exits the parent process. The exec function does not return to the parent process unless the parent calls `vfork` to save the return address.

In *OpenVMS* Version 7.2, the exec functions were enhanced to activate either executable images or DCL command procedures. If no file extension is specified in the *file\_name* argument, the functions first search for the file with the .EXE file extension and then for the file with the .COM file extension. If both the executable image and the command procedure with the same name exist, you must explicitly specify the .COM file extension to force activating the command procedure.

For a DCL command procedure, the exec functions pass the first eight *arg0*, *arg1*, ..., arguments specified in the exec call to the command procedure as *P1*, *P2*, ... parameters, preserving the case.

Unlike UNIX based systems, the exec functions in the *Compaq C* RTL cannot always determine if the specified executable image or command procedure exists and can be activated and executed. Therefore, the exec functions might appear to succeed even though the specified file cannot be executed by the child process.

The status of the child process, returned to the parent process, indicates that the error occurred. You can retrieve this error code by using one of the functions from the wait family of functions.

---

### Note

---

The `vfork` and exec functions in the *Compaq C* RTL on *OpenVMS* systems work differently than on UNIX systems:

- On UNIX systems, `vfork` creates a child process, suspends the parent, and starts the child running where the parent left off.
- On *OpenVMS* systems, `vfork` establishes context later used by an exec function, but it is the exec function, not `vfork`, that starts a process running the specified program.

For a programmer, the key differences are:

- On *OpenVMS* systems, code between the call to `vfork` and the call to an exec function runs in the parent process.  
On UNIX systems, this code runs in the child process.
  - On *OpenVMS* systems, the child inherits open file descriptors and so on, at the point where the exec function is called.  
On UNIX systems, this occurs at the point where `vfork` is called.
-

### 5.2.1 exec Processing

The exec functions use the LIB\$SPAWN routine to create the subprocess and activate the child image within the subprocess. This child process inherits the parent's environment, including all defined logical names and command-line interpreter symbols. The exec functions use the logical name VAXC\$EXECMBX to communicate between parent and child; this logical name must not exist outside the context of the parent image.

The exec functions pass the following information to the child:

- The parent's umask value, which specifies whether any access is to be disallowed when a new file is created. For more information about the umask function, see the Reference Section.
- The file-name string associated with each file descriptor and the current position within each file. The child opens the file and calls lseek to position the file to the same location as the parent. If the file is a record file, the child is positioned on a record boundary, regardless of the parent's position within the record. For more information about file descriptors, see Chapter 2. For more information on the lseek function, see the Reference Section.

This information is sent to the child for all descriptors known to the parent including all descriptors for open files, null descriptors, and duplicate descriptors.

File pointers are not transferred to the child. For files opened by a file pointer in the parent, only their corresponding file descriptors are passed to the child. The fdopen function must be called to associate a file pointer with the file descriptor if the child will access the file-by-file pointer. For more information about the fdopen function, see the Reference Section.

- The signal database. Only SIG\_IGN (ignore) actions are inherited. Actions specified as routines are changed to SIG\_DFL (default) because the parent's signal-handling routines are inaccessible to the child.
- The environment and argument vectors.

When everything is transmitted to the child, exec processing is complete. Control in the parent process then returns to the address saved by vfork and the child's process ID is returned to the parent.

See Section 4.2.4 for a discussion of signal actions and the SIGCHLD signal.

### 5.2.2 exec Error Conditions

The exec functions will fail if LIB\$SPAWN cannot create the subprocess. Conditions that can cause a failure include exceeding the subprocess quota or finding the communications by the context mailbox between the parent and child to be broken. Exceeding some quotas will not cause LIB\$SPAWN to fail, but will put LIB\$SPAWN into a wait state that can cause the parent process to hang. An example of such a quota is the Open File Limit quota.

You will need an Open File Limit quota of at least 20 files, with an average of three times the number of concurrent processes that your program will run. If you use more than one open pipe at a time, or perform I/O on several files at one time, this quota may need to be even higher. See your system manager if this quota needs to be increased.

When an exec function fails, a value of -1 is returned. After such a failure, the parent is expected to call either the exit or \_exit function. Both functions then return to the parent's vfork call, which returns the child's process ID. In this case, the child process ID returned by the exec function is less than zero. For more information about the exit function, see the Reference Section.

## 5.3 Synchronizing Processes

A child process is terminated when the parent process terminates. Therefore, the parent process must check the status of its child processes before exiting. This is done using the *Compaq C* RTL function wait.

## 5.4 Interprocess Communication

A channel through which parent and child processes communicate is called a *pipe*. Use the pipe function to create a pipe.

## 5.5 Program Examples

Example 5-1 shows the basic procedures for executing an image in a child process. The child process in Example 5-1 prints a message 10 times.

### Example 5-1 Creating the Child Process

```
/* chap_5_exec_image.c */

/* This example creates the child process. The only */
/* functionality given to the child is the ability to */
/* print a message 10 times. */

#include <climsgdef.h> /* CLI status values */
#include <stdio.h>
#include <perror.h>
#include <processes.h>
#include <stdlib.h>

static const char *child_name = "chap_5_exec_image_child.exe" ;

main()
{
 int status,
 cstatus;

 /* NOTE: */
 /* Any local automatic variables, even those */
 /* having the volatile attribute, may have */
 /* indeterminate values if they are modified */
 /* between the vfork() call and the matching */
 /* exec() call. */
}
```

(continued on next page)

### Example 5–1 (Cont.) Creating the Child Process

```
❶ if ((status = vfork()) != 0) {
 /* This is either an error or */
 /* the "second" vfork return, taking us "back" */
 /* to parent mode. */
❸ if (status < 0)
 printf("Parent - Child process failed\n");
 else {
 printf("Parent - Waiting for Child\n");
❷ if ((status = wait(&cstatus)) == -1)
 perror("Parent - Wait failed");
❹ else if (cstatus == CLI$_IMAGEFNF)
 printf("Parent - Child does not exist\n");
❺ else
 printf("Parent - Child final status: %d\n", cstatus);
 }
}
else { /* The FIRST Vfork return is zero, do the exec */
 printf("Parent - Starting Child\n");
 if ((status = execl(child name, 0)) == -1) {
 perror("Parent - Execl failed");
 exit(EXIT_FAILURE);
 }
}
}

/* CHAP_5_EXEC_IMAGE_CHILD.C */
/* This is the child program that writes a message */
/* through the parent to "stdout" */
#include <stdio.h>
main()
{
 int i;
 for (i = 0; i < 10; i++)
 printf("Child - executing\n");
 return (255) ; /* Set an unusual success stat */
}
```

Key to Example 5–1:

- ❶ The vfork function is called to set up the return address for the exec call. The vfork function is normally used in the expression of an if statement. This construct allows you to take advantage of the double return aspect of vfork, since one return value is 0 and the other is nonzero.
- ❷ A 0 return value is returned the first time vfork is called and the parent executes the else clause associated with the vfork call, which calls execl.
- ❸ A negative child process ID is returned when an exec function fails. The return value is checked for these conditions.
- ❹ The wait function is used to synchronize the parent and child processes.
- ❺ Since the exec functions can indicate success up to this point even if the image to be activated in the child does not exist, the parent checks the child's return status for the predefined status, CLI\$\_IMAGEFNF (file not found).

In Example 5–2, the parent passes arguments to the child process.

### Example 5–2 Passing Arguments to the Child Process

```
/* CHAP_5_CHILDARG.C */
/* In this example, the arguments are placed in an array, gargv, */
/* but they can be passed to the child explicitly as a zero- */
/* terminated series of character strings. The child program in this */
/* example writes the arguments that have been passed it to stdout. */

#include <climsgdef.h>
#include <stdio.h>
#include <stdlib.h>
#include <perror.h>
#include <processes.h>

const char *child_name = "chap_5_childarg_child.exe" ;

main()
{
 int status,
 cstatus;
 char *gargv[] =
 {"Child", "ARGC1", "ARGC2", "Parent", 0};
 if ((status = vfork()) != 0) {
 if (status < -1)
 printf("Parent - Child process failed\n");
 else {
 printf("Parent - waiting for Child\n");
 if ((status = wait(&cstatus)) == -1)
 perror("Parent - Wait failed");
 else if (cstatus == CLI$ IMAGEFNF)
 printf("Parent - Child does not exist\n");
 else
 printf("Parent - Child final status: %x\n",
 cstatus);
 }
 }
 else {
 printf("Parent - Starting Child\n");
 if ((status = execv(child_name, gargv)) == -1) {
 perror("Parent - Exec failed");
 exit(EXIT_FAILURE);
 }
 }
}

/* CHAP_5_CHILDARG_CHILD.C */
/* This is a child program that echos its arguments */
#include <stdio.h>

main(argc, argv)
 int argc;
 char *argv[];
{
 int i;
 printf("Program name: %s\n", argv[0]);
}
```

(continued on next page)

### Example 5–2 (Cont.) Passing Arguments to the Child Process

```
 for (i = 1; i < argc; i++)
 printf("Argument %d: %s\n", i, argv[i]);
 return(255) ;
}
```

Example 5–3 shows how to use the wait function to check the final status of multiple children being run simultaneously.

### Example 5–3 Checking the Status of Child Processes

```
/* CHAP_5_CHECK_STAT.C */
/* In this example 5 child processes are started. The wait() */
/* function is placed in a separate for loop so that it is */
/* called once for each child. If wait() were called within */
/* the first for loop, the parent would wait for one child to */
/* terminate before executing the next child. If there were */
/* only one wait request, any child still running when the */
/* parent exits would terminate prematurely. */
#include <climsgdef.h>
#include <stdio.h>
#include <stdlib.h>
#include <perror.h>
#include <processes.h>

const char *child_name = "chap_5_check_stat_child.exe" ;

main()
{
 int status,
 cstatus,
 i;

 for (i = 0; i < 5; i++) {
 if ((status = vfork()) == 0) {
 printf("Parent - Starting Child %d\n", i);
 if ((status = execl(child_name, 0)) == -1) {
 perror("Parent - Exec failed");
 exit(EXIT_FAILURE);
 }
 }
 else if (status < -1)
 printf("Parent - Child process failed\n");
 }

 printf("Parent - Waiting for children\n");

 for (i = 0; i < 5; i++) {
 if ((status = wait(&cstatus)) == -1)
 perror("Parent - Wait failed");
 else if (cstatus == CLI$ IMAGEFNF)
 printf("Parent - Child does not exist\n");
 else
 printf("Parent - Child %X final status: %d\n",
 status, cstatus);
 }
}
```



Example 5–4 shows how to use the pipe and dup2 functions to communicate between a parent and child process through specific file descriptors. The #define preprocessor directive defines the preprocessor constants inpipe and outpipe as the names of file descriptors 11 and 12.

#### Example 5–4 Communicating Through a Pipe

```
/* CHAP_5_PIPE.C */
/* In this example, the parent writes a string to the pipe for */
/* the child to read. The child then writes the string back */
/* to the pipe for the parent to read. The wait function is */
/* called before the parent reads the string that the child has */
/* passed back through the pipe. Otherwise, the reads and */
/* writes will not be synchronized. */

#include <perror.h>
#include <climits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <processes.h>
#include <unistd.h>

#define inpipe 11
#define outpipe 12

const char *child_name = "chap_5_pipe_child.exe" ;

main()
{
 int pipes[2];
 int mode,
 status,
 cstatus,
 len;
 char *outbuf,
 *inbuf;

 if ((outbuf = malloc(512)) == 0) {
 printf("Parent - Outbuf allocation failed\n");
 exit(EXIT_FAILURE);
 }

 if ((inbuf = malloc(512)) == 0) {
 printf("Parent - Inbuf allocation failed\n");
 exit(EXIT_FAILURE);
 }

 if (pipe(pipes) == -1) {
 printf("Parent - Pipe allocation failed\n");
 exit(EXIT_FAILURE);
 }

 dup2(pipes[0], inpipe);
 dup2(pipes[1], outpipe);
 strcpy(outbuf, "This is a test of two-way pipes.\n");
 status = vfork();
```

(continued on next page)

#### Example 5–4 (Cont.) Communicating Through a Pipe

```
switch (status) {
case 0:
 printf("Parent - Starting child\n");
 if ((status = execl(child_name, 0)) == -1) {
 printf("Parent - Exec failed");
 exit(EXIT_FAILURE);
 }
 break;
case -1:
 printf("Parent - Child process failed\n");
 break;
default:
 printf("Parent - Writing to child\n");
 if (write(outpipe, outbuf, strlen(outbuf) + 1) == -1) {
 perror("Parent - Write failed");
 exit(EXIT_FAILURE);
 }
 else {
 if ((status = wait(&cstatus)) == -1)
 perror("Parent - Wait failed");
 if (cstatus == CLIS$ IMAGEFNF)
 printf("Parent - Child does not exist\n");
 else {
 printf("Parent - Reading from child\n");
 if ((len = read(inpipe, inbuf, 512)) <= 0) {
 perror("Parent - Read failed");
 exit(EXIT_FAILURE);
 }
 else {
 printf("Parent: %s\n", inbuf);
 printf("Parent - Child final status: %d\n",
 cstatus);
 }
 }
 }
 break;
}
}
```

---

```
/* CHAP_5_PIPE_CHILD.C */
/* This is a child program which reads from a pipe and writes */
/* the received message back to its parent. */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#define inpipe 11
#define outpipe 12
main()
{
 char *buffer;
 int len;
```

(continued on next page)

#### Example 5–4 (Cont.) Communicating Through a Pipe

```
if ((buffer = malloc(512)) == 0) {
 perror("Child - Buffer allocation failed\n");
 exit(EXIT_FAILURE);
}

printf("Child - Reading from parent\n");
if ((len = read(inpipe, buffer, 512)) <= 0) {
 perror("Child - Read failed");
 exit(EXIT_FAILURE);
}
else {
 printf("Child: %s\n", buffer);
 printf("Child - Writing to parent\n");
 if (write(outpipe, buffer, strlen(buffer) + 1) == -1) {
 perror("Child - Write failed");
 exit(EXIT_FAILURE);
 }
}
exit(EXIT_SUCCESS);
}
```



---

## Curses Screen Management Functions and Macros

This chapter describes the screen management routines available with *Compaq C* for *OpenVMS* Systems.

On all *OpenVMS* systems, the *OpenVMS* Curses screen management package is supported. This is the same package of routines used by the VAX C Run-Time Library (RTL).

On *OpenVMS Alpha* systems, two screen management packages are supported: *OpenVMS* Curses and a more UNIX compatible package based on the Berkeley Standard Distribution (BSD) Curses software.<sup>1</sup> See Section 6.1 for more information.

Furthermore, beginning with *OpenVMS Alpha* Version 7.0, the *Compaq C* RTL offers a Curses package based on the 4.4BSD Berkeley Software Distribution. Documentation on the 4.4BSD Curses package can be found in *Screen Updating and Cursor Movement Optimization: A Library Package*, by Kenneth C.R.C. Arnold.

The functions and macros in the *OpenVMS* and BSD-based Curses packages are nearly the same. Most differences between them are called out in this chapter. Otherwise, this chapter makes no distinction between the two Curses packages, and refers to "Curses" or the "Curses functions and macros."

### 6.1 Using the BSD-Based Curses Package *(Alpha only)*

The `<curses.h>` header file required to use the BSD-based Curses implementation is provided with the *Compaq C* compiler on *OpenVMS Alpha* systems.

Existing programs are not affected by the BSD-based Curses functions because the *OpenVMS* Curses functions are still available as the default Curses package. (Note that is a change from previous versions of *Compaq C*, where BSD-based Curses was the default.)

To get the the 4.4BSD Curses implementation, you must compile modules that include `<curses.h>` with the following qualifier:

```
/DEFINE=_BSD44_CURSES
```

The BSD-based Curses functions do not provide the support required to call the *OpenVMS* SMG\$ routines with the pasteboard and keyboard allocated by the Curses functions. Consequently, Curses programs that rely on calling SMG\$ entry points, as well as Curses functions, must continue to use the *OpenVMS* Curses implementation.

---

<sup>1</sup> Copyright (c) 1981 Regents of the University of California.  
All rights reserved.

The BSD-based Curses implementation is not interoperable with the old implementation. Attempts to mix calls to the new functions and the old functions will result in incorrect output displayed on the screen and could result in an exception from an SMG\$ routine.

## 6.2 Curses Overview

Curses, the *Compaq C* Screen Management Package, is composed of *Compaq C* RTL functions and macros that create and modify defined sections of the terminal screen and optimize cursor movement. Using the screen management package, you can develop a user interface that is both visually attractive and user-friendly. Curses is terminal-independent and provides simplified terminal screen formatting and efficient cursor movement.

Most Curses functions and macros are listed in pairs where the first routine is a macro and the second is a function beginning with the prefix “w,” for “window.” These prefixes are delimited by brackets ([ ]). For example, [w]addstr designates the addstr macro and the waddstr function. The macros default to the window stdscr; the functions accept a specified window as an argument.

To access the Curses functions and macros, include the <curses.h> header file.

The terminal-independent Screen Management Software, which is part of the *OpenVMS* RTL, is used to implement Curses. For portability purposes, most functions and macros are designed to perform in a manner similar to other C implementations. However, the Curses routines depend on the *OpenVMS* system and its Screen Management Software, so performance of some functions and macros could differ slightly from those of other implementations.

Some functions and macros available on other systems are not available with the *Compaq C* RTL Curses package.

Some functions, such as [w]clrattr, [w]insstr, mv[w]insstr, and [w]setattr are specific to *Compaq C* for *OpenVMS* Systems and are not portable.

Table 6–1 lists all of the Curses functions and macros found in the *Compaq C* RTL. For more detailed information on each function and macro, see the Reference Section.

**Table 6–1 Curses Functions and Macros**

| Function or Macro | Description                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------------------|
| [w]addch          | Adds a character to the window at the current position of the cursor.                                  |
| [w]addstr         | Adds a string to the window at the current position of the cursor.                                     |
| box               | Draws a box around the window.                                                                         |
| [w]clear          | Erases the contents of the specified window and resets the cursor to coordinates (0,0).                |
| clearok           | Sets the clear flag for the window.                                                                    |
| [w]clrattr        | Deactivates the video display attribute within the window.                                             |
| [w]clrtoBot       | Erases the contents of the window from the current position of the cursor to the bottom of the window. |

(continued on next page)

**Table 6–1 (Cont.) Curses Functions and Macros**

| Function or Macro | Description                                                                                                               |
|-------------------|---------------------------------------------------------------------------------------------------------------------------|
| [w]clrtoeol       | Erases the contents of the window from the current cursor position to the end of the line on the specified window.        |
| [no]crmode        | Sets and unsets the terminal from cbreak mode.                                                                            |
| [w]delch          | Deletes the character on the specified window at the current position of the cursor.                                      |
| [w]deleteln       | Deletes the line at the current position of the cursor.                                                                   |
| delwin            | Deletes the specified window from memory.                                                                                 |
| [no]echo          | Sets the terminal so that characters may or may not be echoed on the terminal screen.                                     |
| endwin            | Clears the terminal screen and frees any virtual memory allocated to Curses data structures.                              |
| [w]erase          | Erases the window by painting it with blanks.                                                                             |
| [w]getch          | Gets a character from the terminal screen and echoes it on the specified window.                                          |
| [w]getstr         | Gets a string from the terminal screen, stores it in a character variable, and echoes it on the specified window.         |
| getyx             | Puts the (y,x) coordinates of the current cursor position on the window in the variables y and x.                         |
| [w]inch           | Returns the character at the current cursor position on the specified window without making changes to the window.        |
| initscr           | Initializes the terminal-type data and all screen functions.                                                              |
| [w]insch          | Inserts a character at the current cursor position in the specified window.                                               |
| [w]insertln       | Inserts a line above the line containing the current cursor position.                                                     |
| [w]insstr         | Inserts a string at the current cursor position on the specified window.                                                  |
| leaveok           | Leaves the cursor at the current coordinates after an update to the window.                                               |
| longname          | Assigns the full terminal name to a character name that must be large enough to hold the character string.                |
| [w]move           | Changes the current cursor position on the specified window.                                                              |
| mv[w]addch        | Moves the cursor and adds a character to the specified window.                                                            |
| mv[w]addstr       | Moves the cursor and adds a string to the specified window.                                                               |
| mvcur             | Moves the terminal's cursor.                                                                                              |
| mv[w]delch        | Moves the cursor and deletes a character on the specified window.                                                         |
| mv[w]getch        | Moves the cursor, gets a character from the terminal screen, and echoes it on the specified window.                       |
| mv[w]getstr       | Moves the cursor, gets a string from the terminal screen, stores it in a variable, and echoes it on the specified window. |
| mv[w]inch         | Moves the cursor and returns the character on the specified window without making changes to the window.                  |

(continued on next page)

**Table 6–1 (Cont.) Curses Functions and Macros**

| Function or Macro        | Description                                                                                                                                           |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>mv[w]insch</code>  | Moves the cursor and inserts a character in the specified window.                                                                                     |
| <code>mv[w]insstr</code> | Moves the cursor and inserts a string in the specified window.                                                                                        |
| <code>mvwin</code>       | Moves the starting position of the window to the specified coordinates.                                                                               |
| <code>newwin</code>      | Creates a new window with lines and columns starting at the coordinates on the terminal screen.                                                       |
| <code>[no]nl</code>      | Provided only for UNIX software compatibility and has no functionality in the <i>OpenVMS</i> environment.                                             |
| <code>overlay</code>     | Writes the contents of one window that will fit over the contents of another window, beginning at the starting coordinates of both windows.           |
| <code>overwrite</code>   | Writes the contents of one window, insofar as it will fit, over the contents of another window beginning at the starting coordinates of both windows. |
| <code>[w]printw</code>   | Performs a <code>printf</code> on the window starting at the current position of the cursor.                                                          |
| <code>[no]raw</code>     | Provided only for UNIX software compatibility and has no functionality in the <i>OpenVMS</i> environment.                                             |
| <code>[w]refresh</code>  | Repaints the specified window on the terminal screen.                                                                                                 |
| <code>[w]scanw</code>    | Performs a <code>scanf</code> on the window.                                                                                                          |
| <code>scroll</code>      | Moves all the lines on the window up one line.                                                                                                        |
| <code>scrollok</code>    | Sets the scroll flag for the specified window.                                                                                                        |
| <code>[w]setattr</code>  | Activates the video display attribute within the window.                                                                                              |
| <code>[w]standend</code> | Deactivates the boldface attribute for the specified window.                                                                                          |
| <code>[w]standout</code> | Activates the boldface attribute of the specified window.                                                                                             |
| <code>subwin</code>      | Creates a new subwindow with lines and columns starting at the coordinates on the terminal screen.                                                    |
| <code>touchwin</code>    | Places the most recently edited version of the specified window on the terminal screen.                                                               |
| <code>wrapok</code>      | <i>OpenVMS</i> Curses only. Allows the wrapping of a word from the right border of the window to the beginning of the next line.                      |

## 6.3 Curses Terminology

This section explains some of the Curses terminology and shows you how Curses looks on the terminal screen.

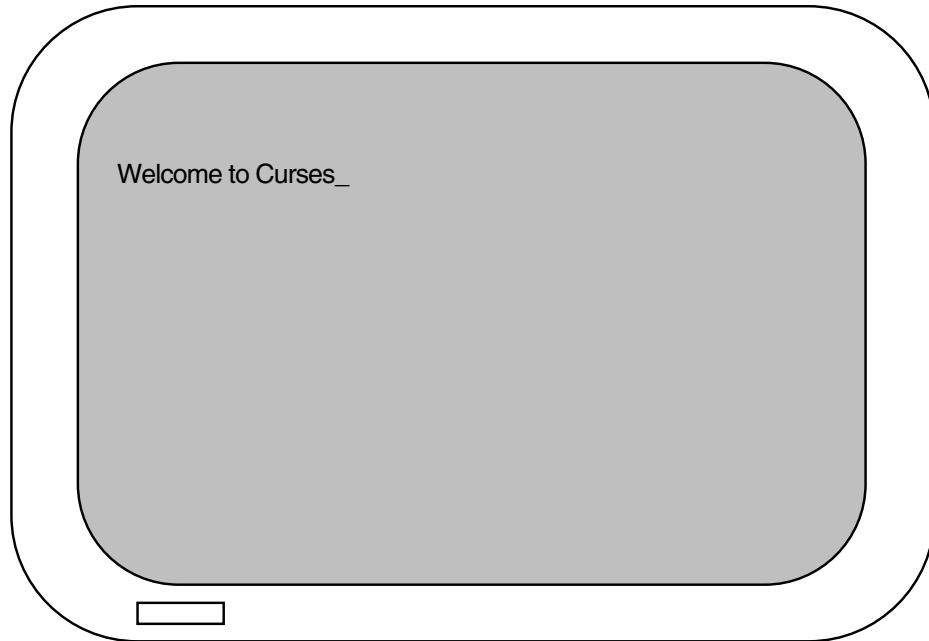
Consider a Curses application as being a series of overlapping windows. Window overlapping is called *occlusion*. To distinguish the boundaries of these occluding windows, you can outline the rectangular windows with specified characters, or you can turn on the reverse video option (make the window a light background with dark writing).



### 6.3.1 Predefined Windows (stdscr and curscr)

Initially, two windows the size of the terminal screen are predefined by Curses. These windows are called `stdscr` and `curscr`. The `stdscr` window is defined for your use. Many Curses macros default to this window. For example, if you draw a box around `stdscr`, move the cursor to the left-corner area of the screen, write a string to `stdscr`, and then display `stdscr` on the terminal screen, your display will look like that in Figure 6–1.

**Figure 6–1 An Example of the `stdscr` Window**



ZK-5752-GE

The second predefined window, `curscr`, is designed for internal Curses work; it is an image of what is currently displayed on the terminal screen. The only *Compaq C* for *OpenVMS* Curses function that will accept this window as an argument is `clearok`. Do not write to or read from `curscr`. Use `stdscr` and user-defined windows for all your Curses applications.

### 6.3.2 User-Defined Windows

You can occlude `stdscr` with your own windows. The size and location of each window is given in terms of the number of lines, the number of columns, and the starting position.

The lines and columns of the terminal screen form a coordinate system, or grid, on which the windows are formed. You specify the starting position of a window with the (y,x) coordinates on the terminal screen where the upper left corner of the window is located. The coordinates (0,0) on the terminal screen, for example, are the upper left corner of the screen.

The entire area of the window must be within the terminal screen borders; windows can be as small as a single character or as large as the entire terminal screen. You can create as many windows as memory allows.

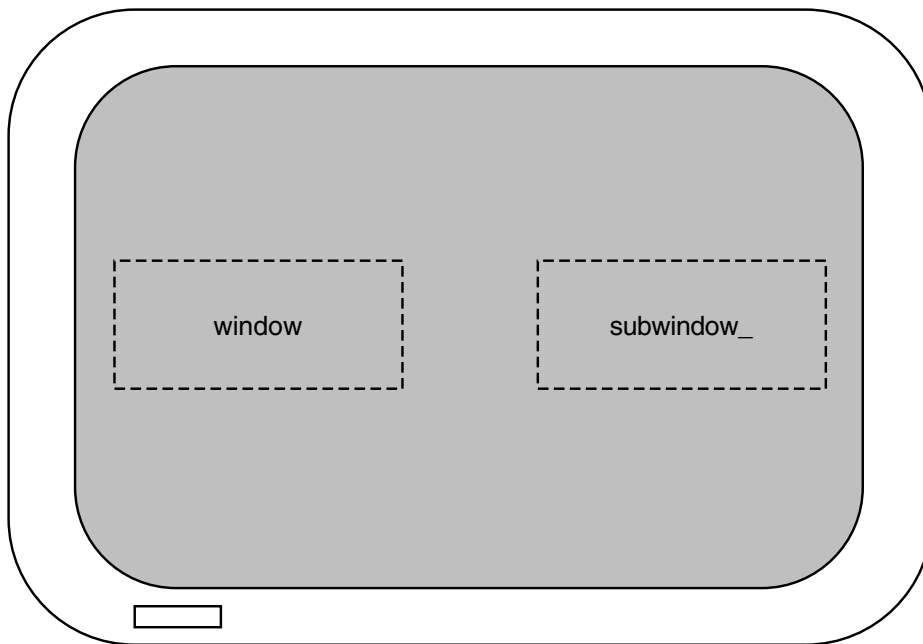
When writing to or deleting from windows, changes do not appear on the terminal screen until the window is *refreshed*. When refreshing a window, you place the updated window onto the terminal screen, which leaves the rest of the screen unaltered.

All user-defined windows, by default, occlude `stdscr`. You can create two or more windows that occlude each other as well as `stdscr`. When writing data to one occluding window, the data is not written to the underlying window.

You can create overlapping windows (called *subwindows*). A declared window must contain the entire area of its subwindow. When writing data to a subwindow or to the portion of the window overlapped by the subwindow, both windows contain the new data. For instance, if you write data to a subwindow and then delete that subwindow, the data is still present on the underlying window.

If you create a window that occludes `stdscr` and a subwindow of `stdscr`, your terminal screen will look like Figure 6–2.

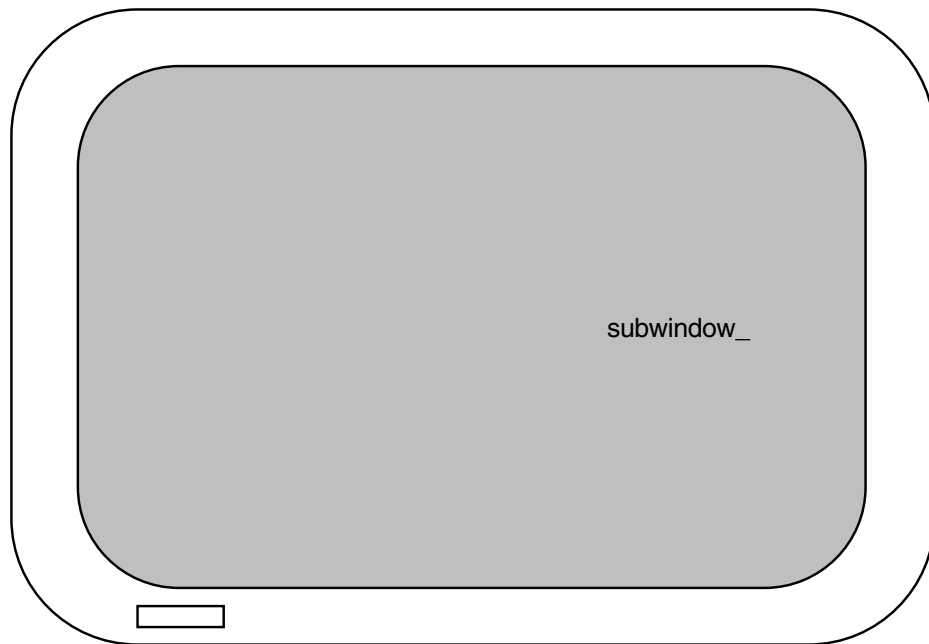
**Figure 6–2 Displaying Windows and Subwindows**



ZK-5754-GE

If you delete both the user-defined window and the subwindow, and then update the terminal screen with the new image, your terminal screen will look like Figure 6–3.

**Figure 6–3 Updating the Terminal Screen**



ZK-5753-GE

The string written on the window is deleted, but the string written on the subwindow remains on stdscr.

## 6.4 Getting Started with Curses

There are commands that you must use to initialize and restore the terminal screen when using Curses Screen Management functions and macros. Also, there are predefined variables and constants on which Curses depends. Example 6–1 shows how to set up a program using Curses.

### Example 6–1 A Curses Program

```
❶ #include <curses.h>
❷ WINDOW *win1, *win2, *win3;

main()
{
❸ initscr();
 .
 .
 .
 endwin();
}
```

Key to Example 6–1:

- ❶ The preprocessor directive includes the `<curses.h>` header file, which defines the data structures and variables used to implement Curses. The `<curses.h>` header file includes the `<stdio.h>` header file, so it is not necessary to duplicate this action by including `<stdio.h>` again in the program source

code. You must include `<curses.h>` to use any of the Curses functions or macros.

- ❸ In the example, `WINDOW` is a data structure defined in `<curses.h>`. You must declare each user-specified window in this manner. In Example 6–1, the three defined windows are `win1`, `win2`, and `win3`.
- ❹ The `initscr` and `endwin` functions begin and end the window editing session. The `initscr` function clears the terminal screen (*OpenVMS* Curses only; BSD-based Curses does not), and allocates space for the windows `stdscr` and `curscr`. The `endwin` function deletes all windows and clears the terminal screen.

Most Curses users wish to define and modify windows. Example 6–2 shows you how to define and write to a single window.

### Example 6–2 Manipulating Windows

```
#include <curses.h>
WINDOW *win1, *win2, *win3;
main()
{
 initscr();
 ❶ win1 = newwin(24, 80, 0, 0);
 ❷ mvwaddstr(win1, 2, 2, "HELLO");
 .
 .
 .
 endwin();
}
```

Key to Example 6–2:

- ❶ The `newwin` function defines a window 24 rows high and 80 columns wide with a starting position at coordinates (0,0), the upper left corner of the terminal screen. The program assigns these attributes to `win1`. The coordinates are specified as follows: (lines,columns) or (y,x).
- ❷ The `mvwaddstr` macro performs the same task as a call to the separate macros `move` and `addstr`. The `mvwaddstr` macro moves the cursor to the specified coordinates and writes a string onto `stdscr`.

---

#### Note

---

Most Curses macros update `stdscr` by default. Curses functions that update other windows have the same name as the macros but with the added prefix “w”. For example, the `addstr` macro adds a given string to `stdscr` at the current cursor position. The `waddstr` function adds a given string to a specified window at the current cursor position.

---

When updating a window, specify the cursor position relative to the origin of the window, not the origin of the terminal screen. For example, if a window has a starting position of (10,10) and you want to add a character to the window at its starting position, specify the coordinates (0,0), not (10,10).

The string HELLO in Example 6–2 does not appear on the terminal screen until you refresh the screen. You accomplish this by using the wrefresh function. Example 6–3 shows how to display the contents of win1 on the terminal screen.

**Example 6–3 Refreshing the Terminal Screen**

```
#include <curses.h>
WINDOW *win1, *win2, *win3;
main()
{
 initscr();
 win1 = newwin(22, 60, 0, 0);
 mvwaddstr(win1, 2, 2, "HELLO");
 wrefresh(win1);
 .
 .
 .
 endwin();
}
```

The wrefresh function updates just the region of the specified window on the terminal screen. When the program is executed, the string HELLO appears on the terminal screen until the program executes the endwin function. The wrefresh function only refreshes the part of the window on the terminal screen that is not overlapped by another window. If win1 was overlapped by another window and you want all of win1 to be displayed on the terminal screen, call the touchwin function.

**6.5 Predefined Variables and Constants**

The <curses.h> header file defines variables and constants useful for implementing Curses (see Table 6–2).

**Table 6–2 Curses Predefined Variables and #define Constants**

| Name   | Type     | Description                              |
|--------|----------|------------------------------------------|
| curscr | WINDOW * | Window of current screen                 |
| stdscr | WINDOW * | Default window                           |
| LINES  | int      | Number of lines on the terminal screen   |
| COLS   | int      | Number of columns on the terminal screen |
| ERR    | —        | Flag (0) for failed routines             |
| OK     | —        | Flag (1) for successful routines         |
| TRUE   | —        | Boolean true flag (1)                    |
| FALSE  | —        | Boolean false flag (0)                   |
| _BLINK | —        | Parameter for setattr and clrattr        |
| _BOLD  | —        | Parameter for setattr and clrattr        |

(continued on next page)

**Table 6–2 (Cont.) Curses Predefined Variables and #define Constants**

| Name                    | Type | Description                                                 |
|-------------------------|------|-------------------------------------------------------------|
| <code>_REVERSE</code>   | —    | Parameter for <code>setattr</code> and <code>clrattr</code> |
| <code>_UNDERLINE</code> | —    | Parameter for <code>setattr</code> and <code>clrattr</code> |

For example, you can use the predefined macro `ERR` to test the success or failure of a Curses function. Example 6–4 shows how to perform such a test.

**Example 6–4 Curses Predefined Variables**

```
#include <curses.h>
WINDOW *win1, *win2, *win3;
main()
{
 initscr();
 win1 = newwin(10, 10, 1, 5);
 .
 .
 .
 if (mvwin(win1, 1, 10) == ERR)
 addstr("The MVWIN function failed.");
 .
 .
 .
 endwin();
}
```

In Example 6–4, if the `mvwin` function fails, the program adds a string to `stdscr` that explains the outcome. The Curses `mvwin` function moves the starting position of a window.

## 6.6 Cursor Movement

In the UNIX system environment, you can use Curses functions to move the cursor across the terminal screen. With other implementations, you can either allow Curses to move the cursor using the `move` function, or you can specify the origin and the destination of the cursor to the `mvcur` function, which moves the cursor in a more efficient manner.

In *Compaq C for OpenVMS Systems*, the two functions are functionally equivalent and move the cursor with the same efficiency.

Example 6–5 shows how to use the `move` and `mvcur` functions.

### Example 6–5 The Cursor Movement Functions

```
#include <curses.h>

main()
{
 initscr();
 .
 .
 .
 ❶ clear();
 ❷ move(10, 10);
 ❸ move(LINES/2, COLS/2);
 ❹ mvcur(0, COLS-1, LINES-1, 0);
 .
 .
 .
 endwin();
}
```

Key to Example 6–5:

- ❶ The clear macro erases stdscr and positions the cursor at coordinates (0,0).
- ❷ The first occurrence of move moves the cursor to coordinates (10,10).
- ❸ The second occurrence of move uses the predefined variables LINES and COLS to calculate the center of the screen (by calculating the value of half the number of LINES and COLS on the screen).
- ❹ The mvcur function forces absolute addressing. This function can address the lower left corner of the screen by claiming that the cursor is presently in the upper right corner. You can use this method if you are unsure of the current position of the cursor, but move works just as well.

## 6.7 Program Example

The following program example shows the effects of many of the Curses macros and functions. You can find explanations of the individual lines of code, if not self-explanatory, in the comments to the right of the particular line. Detailed discussions of the functions follow the source code listing.

Example 6–6 shows the definition and manipulation of one user-defined window and stdscr.

### Example 6–6 stdscr and Occluding Windows

```
/* CHAP_6_STDSCR_OCCLUDE.C */
/* This program defines one window: win1. win1 is */
/* located towards the center of the default window */
/* stdscr. When writing to an occluding window (win1) */
/* that is later erased, the writing is erased as well. */
#include <curses.h> /* Include header file. */
WINDOW *win1; /* Define windows. */
main()
{
 char str[80]; /* Variable declaration.*/
```

(continued on next page)

### Example 6–6 (Cont.) stdscr and Occluding Windows

```
 initscr(); /* Set up Curses. */
 noecho(); /* Turn off echo. */

 /* Create window. */
 winl = newwin(10, 20, 10, 10);

 box(stdscr, '|', '-'); /* Draw a box around stdscr. */
 box(winl, '|', '-'); /* Draw a box around winl. */

 refresh(); /* Display stdscr on screen. */
 wrefresh(winl); /* Display winl on screen. */

❶ getstr(str); /* Pause. Type a few words! */

❷ mvaddstr(22, 1, str);
 getch();
 /* Add string to winl. */

 mvwaddstr(winl, 5, 5, "Hello");
 wrefresh(winl); /* Add winl to terminal scr. */

 getch(); /* Pause. Press Return. */

 delwin(winl); /* Delete winl. */

❸ touchwin(stdscr); /* Refresh all of stdscr. */

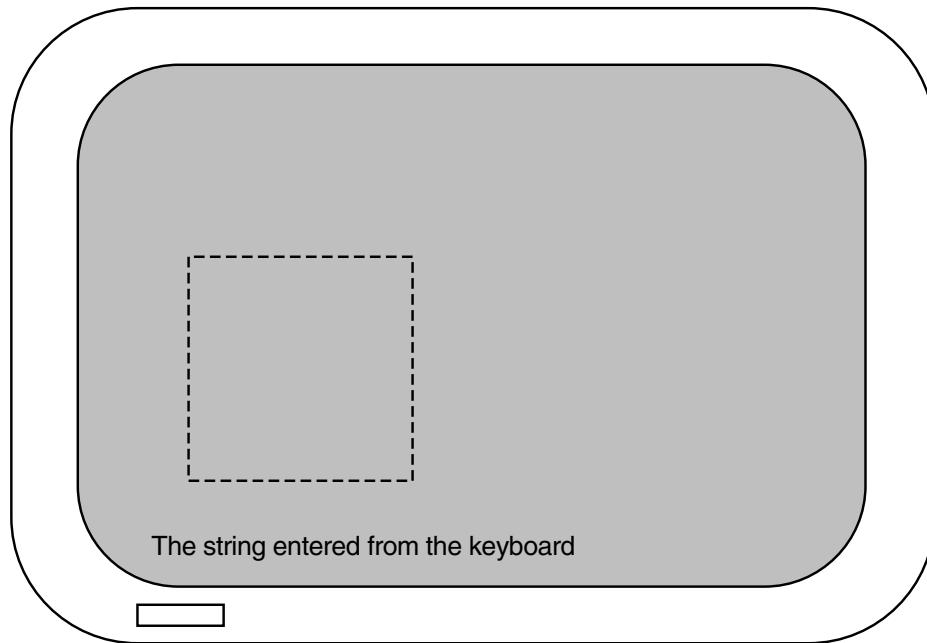
 getch(); /* Pause. Press Return. */
 endwin(); /* Ends session. */
}
```

Key to Example 6–6:

- ❶ The program waits for input. The echo was disabled using the `noecho` macro, so the words that you type do not appear on `stdscr`. However, the macro stores the words in the variable `str` for use elsewhere in the program.
- ❷ The `getch` macro causes the program to pause. When you are finished viewing the screen, press Return so the program can resume. The `getch` macro refreshes `stdscr` on the terminal screen without calling `refresh`. The screen appears like Figure 6–4.



**Figure 6–4 An Example of the getch Macro**



ZK-5751-GE

- ③ The touchwin function refreshes the screen so that all of stdscr is visible and the deleted occluding window no longer appears on the screen.



## Math Functions

Table 7–1 lists and describes the math functions in the *Compaq C* Run-Time Library (RTL). For more detailed information on each function, see the Reference Section.

**Table 7–1 Math Functions**

| Function                                    | Description                                                                                      |
|---------------------------------------------|--------------------------------------------------------------------------------------------------|
| <code>abs</code>                            | Returns the absolute value of an integer.                                                        |
| <code>acos</code>                           | Returns the arc cosine of its radian argument, in the range $[0, \pi]$ radians.                  |
| <code>acosd</code> ( <i>Alpha only</i> )    | Returns the arc cosine of its radian argument, in the range $[0, 180]$ degrees.                  |
| <code>acosh</code> ( <i>Alpha only</i> )    | Returns the hyperbolic arc cosine of its argument.                                               |
| <code>asin</code>                           | Returns the arc sine of its radian argument in the range $[-\pi/2, \pi/2]$ radians.              |
| <code>asind</code> ( <i>Alpha only</i> )    | Returns the arc sine of its radian argument, in the range $[-90, 90]$ degrees.                   |
| <code>asinh</code> ( <i>Alpha only</i> )    | Returns the hyperbolic arc sine of its argument.                                                 |
| <code>atan</code>                           | Returns the arc tangent of its radian argument, in the range $[-\pi/2, \pi/2]$ radians.          |
| <code>atand</code> ( <i>Alpha only</i> )    | Returns the arc tangent of its radian argument, in the range $[-90, 90]$ degrees.                |
| <code>atan2</code>                          | Returns the arc tangent of $y/x$ (its two radian arguments), in the range $[-\pi, \pi]$ radians. |
| <code>atand2</code> ( <i>Alpha only</i> )   | Returns the arc tangent of $y/x$ (its two radian arguments), in the range $[-180, 180]$ degrees. |
| <code>atanh</code> ( <i>Alpha only</i> )    | Returns the hyperbolic arc tangent of its radian argument.                                       |
| <code>cabs</code>                           | Returns the absolute value of a complex number as: $\text{sqrt}(x^2 + y^2)$ .                    |
| <code>cbrt</code> ( <i>Alpha only</i> )     | Returns the rounded cube root of its argument.                                                   |
| <code>ceil</code>                           | Returns the smallest integer greater than or equal to its argument.                              |
| <code>copysign</code> ( <i>Alpha only</i> ) | Returns its first argument with the same sign as its second.                                     |
| <code>cos</code>                            | Returns the cosine of its radian argument in radians.                                            |
| <code>cosd</code> ( <i>Alpha only</i> )     | Returns the cosine of its radian argument in degrees.                                            |
| <code>cosh</code>                           | Returns the hyperbolic cosine of its argument.                                                   |
| <code>cot</code>                            | Returns the cotangent of its radian argument in radians.                                         |

(continued on next page)

**Table 7–1 (Cont.) Math Functions**

| Function                                                                                                                                     | Description                                                                                                                         |
|----------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <code>cotd</code> <i>(Alpha only)</i>                                                                                                        | Returns the cotangent of its radian argument in degrees.                                                                            |
| <code>drand48</code> , <code>erand48</code> ,<br><code>jrand48</code> , <code>lrand48</code> ,<br><code>mrand48</code> , <code>rand48</code> | Generates uniformly distributed pseudorandom number sequences. Returns 48-bit, nonnegative, double-precision floating-point values. |
| <code>erf</code> <i>(Alpha only)</i>                                                                                                         | Returns the error function of its argument.                                                                                         |
| <code>erfc</code> <i>(Alpha only)</i>                                                                                                        | Returns $(1.0 - \text{erf}(x))$ .                                                                                                   |
| <code>exp</code>                                                                                                                             | Returns the base e raised to the power of the argument.                                                                             |
| <code>expm1</code> <i>(Alpha only)</i>                                                                                                       | Returns $\exp(x) - 1$ .                                                                                                             |
| <code>fabs</code>                                                                                                                            | Returns the absolute value of a floating-point value.                                                                               |
| <code>finite</code> <i>(Alpha only)</i>                                                                                                      | Returns 1 if its argument is a finite number; 0 if not.                                                                             |
| <code>floor</code>                                                                                                                           | Returns the largest integer less than or equal to its argument.                                                                     |
| <code>fmod</code>                                                                                                                            | Computes the floating-point remainder of its first argument divided by its second.                                                  |
| <code>fp_class</code> <i>(Alpha only)</i>                                                                                                    | Determines the class of IEEE floating-point values, returning a constant from the <code>&lt;fp_class.h&gt;</code> header file.      |
| <code>isnan</code> <i>(Alpha only)</i>                                                                                                       | Test for NaN. Returns 1 if its argument is a NaN; 0 if not.                                                                         |
| <code>j0</code> , <code>j1</code> , <code>jn</code> <i>(Alpha only)</i>                                                                      | Computes Bessel functions of the first kind.                                                                                        |
| <code>frexp</code>                                                                                                                           | Calculates the fractional and exponent parts of a floating-point value.                                                             |
| <code>hypot</code>                                                                                                                           | Returns the square root of the sum of the squares of two arguments.                                                                 |
| <code>initstate</code>                                                                                                                       | Initializes random number generators.                                                                                               |
| <code>labs</code>                                                                                                                            | Returns the absolute value of an integer as a long int.                                                                             |
| <code>lcong48</code>                                                                                                                         | Initializes a 48-bit uniformly distributed pseudorandom number sequence.                                                            |
| <code>lgamma</code> <i>(Alpha only)</i>                                                                                                      | Computes the logarithm of the gamma function.                                                                                       |
| <code>llabs</code> , <code>qabs</code> <i>(Alpha only)</i>                                                                                   | Returns the absolute value of an <code>__int64</code> integer.                                                                      |
| <code>ldexp</code>                                                                                                                           | Returns its first argument multiplied by 2 raised to the power of its second argument.                                              |
| <code>ldiv</code> , <code>div</code>                                                                                                         | Returns the quotient and remainder after the division of their arguments.                                                           |
| <code>lldiv</code> , <code>qdiv</code> <i>(Alpha only)</i>                                                                                   | Returns the quotient and remainder after the division of their arguments.                                                           |
| <code>log2</code> <i>(Alpha only)</i> , <code>log</code> ,<br><code>log10</code>                                                             | Returns the logarithm of their arguments.                                                                                           |
| <code>log1p</code> <i>(Alpha only)</i>                                                                                                       | Computes $\ln(1+x)$ accurately.                                                                                                     |
| <code>logb</code> <i>(Alpha only)</i>                                                                                                        | Returns the radix-independent exponent of its argument.                                                                             |
| <code>nextafter</code> <i>(Alpha only)</i>                                                                                                   | Returns the next machine-representable number following $x$ in the direction of $y$ .                                               |
| <code>nint</code> <i>(Alpha only)</i>                                                                                                        | Returns the nearest integral value to the argument.                                                                                 |

(continued on next page)

**Table 7–1 (Cont.) Math Functions**

| Function                                    | Description                                                                                                                                               |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>modf</code>                           | Returns the positive fractional part of its first argument and assigns the integral part to the object whose address is specified by the second argument. |
| <code>pow</code>                            | Returns the first argument raised to the power of the second.                                                                                             |
| <code>rand, srand</code>                    | Returns pseudorandom numbers in the range 0 to $2^{31} - 1$ .                                                                                             |
| <code>random, srandom</code>                | Generates pseudorandom numbers in a more random sequence.                                                                                                 |
| <code>rint</code> <i>(Alpha only)</i>       | Rounds its argument to an integral value according to the current IEEE rounding direction specified by the user.                                          |
| <code>scalb</code> <i>(Alpha only)</i>      | Returns the exponent of a floating-point number.                                                                                                          |
| <code>seed48, srand48</code>                | Initializes a 48-bit random number generator.                                                                                                             |
| <code>setstate</code>                       | Restarts, and changes random number generators.                                                                                                           |
| <code>sin</code>                            | Returns the sine of its radian argument in radians.                                                                                                       |
| <code>sind</code> <i>(Alpha only)</i>       | Returns the sine of its radian argument in degrees.                                                                                                       |
| <code>sinh</code>                           | Returns the hyperbolic sine of its argument.                                                                                                              |
| <code>sqrt</code>                           | Returns the square root of its argument.                                                                                                                  |
| <code>tan</code>                            | Returns the tangent of its radian argument in radians.                                                                                                    |
| <code>tand</code> <i>(Alpha only)</i>       | Returns the tangent of its radian argument in degrees.                                                                                                    |
| <code>tanh</code>                           | Returns the hyperbolic tangent of its argument.                                                                                                           |
| <code>trunc</code> <i>(Alpha only)</i>      | Truncates its argument to an integral value.                                                                                                              |
| <code>unordered</code> <i>(Alpha only)</i>  | Returns 1 if either or both of its arguments is a NaN; 0, if not.                                                                                         |
| <code>y0, y1, yn</code> <i>(Alpha only)</i> | Computes Bessel functions of the second kind.                                                                                                             |

## 7.1 Math Function Variants—float, long double

Additional math routine variants are supported for *Compaq C* on *OpenVMS Alpha* systems only. They are defined in `<math.h>` and are float and long double variants of the routines listed in Table 7–1.

Float variants take float arguments and return float values. Their names have an `f` suffix. For example:

```
float cosf (float x);
float tandf (float x);
```

Long double variants take long double arguments and return long double values. Their names have an `l` suffix. For example:

```
long double cosl (long double x);
long double tandl (long double x);
```

All math routine variants are included in the Reference Section of this manual.

Note that for programs compiled without `/L_DOUBLE=64` (that is, compiled with the default `/L_DOUBLE=128`), the long double variants of these *Compaq C* RTL math routines map to the `X_FLOAT` entry points documented in the *Compaq Portable Mathematics Library* (CPML) manual.

## 7.2 Error Detection

To help you detect run-time errors, the `<errno.h>` header file defines the following two symbolic values that are returned by many (but not all) of the mathematical functions:

- `EDOM` indicates that an argument is inappropriate; the argument is not within the function's domain.
- `ERANGE` indicates that a result is out of range; the argument is too large or too small to be represented by the machine.

When using the math functions, you can check the external variable `errno` for either or both of these values and take the appropriate action if an error occurs.

The following program example checks the variable `errno` for the value `EDOM`, which indicates that a negative number was specified as input to the function `sqrt`:

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

main()
{
 double input, square_root;
 printf("Enter a number: ");
 scanf("%le", &input);
 errno = 0;
 square_root = sqrt(input);

 if (errno == EDOM)
 perror("Input was negative");
 else
 printf("Square root of %e = %e\n",
 input, square_root);
}
```

If you did not check `errno` for this symbolic value, the `sqrt` function returns 0 when a negative number is entered. For more information about the `<errno.h>` header file, see Chapter 4.

## 7.3 The `<fp.h>` Header File

The `<fp.h>` header file implements some of the features defined by the Numerical C Extensions Group of the ANSI X3J11 committee. You might find this useful for applications that make extensive use of floating-point functions.

Some of the double-precision functions listed in this chapter return the value `±HUGE_VAL` (defined in either `<math.h>` or `<fp.h>`) if the result is out of range. The float version of those functions return the value `HUGE_VALF` (defined only in `<fp.h>`) for the same conditions. The long double version returns the value `HUGE_VALL` (also defined in `<fp.h>`).

For programs compiled to enable IEEE infinity and NaN values, the values `HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL` are expressions, not compile-time constants. Initializations such as the following cause a compile-time error:

```

$ CREATE IEEE_INFINITY.C
#include <fp.h>

double my_huge_val = HUGE_VAL
^Z
$ CC /FLOAT=IEEE/IEEE=DENORM IEEE_INFINITY
double my_huge_val = HUGE_VAL;
.....^
%CC-E-NEEDCONSTEXPR, In the initializer for my_huge_val, "decc$gt_dbl_infinity"
is not constant, but occurs in a context that requires a constant expression.
at line number 3 in file WORK1$:[RTL]IEEE_INFINITY.C;1
$

```

When using both `<math.h>` and `<fp.h>`, be aware that `<math.h>` defines a function `isnan` and `<fp.h>` defines a macro by the same name. Whichever header is included first in the application will resolve a reference to `isnan`.

## 7.4 Example

Example 7–1 shows how the `tan`, `sin`, and `cos` functions operate.

### Example 7–1 Calculating and Verifying a Tangent Value

```

/* CHAP_7_MATH_EXAMPLE.C */
/* This example uses two functions --- mytan and main --- */
/* to calculate the tangent value of a number, and to check */
/* the calculation using the sin and cos functions. */

#include <math.h>
#include <stdio.h>

/* This function calculates the tangent using the sin and */
/* cos functions. */

double mytan(x)
double x;
{
 double y,
 y1,
 y2;

 y1 = sin(x);
 y2 = cos(x);

 if (y2 == 0)
 y = 0;
 else
 y = y1 / y2;

 return y;
}

main()
{
 double x;

 /* Print values: compare */
 for (x = 0.0; x < 1.5; x += 0.1)
 printf("tan of %4.1f = %6.2f\t%6.2f\n", x, mytan(x), tan(x));
}

```

Example 7–1 produces the following output:

```
$ RUN EXAMPLE
tan of 0.0 = 0.00 0.00
tan of 0.1 = 0.10 0.10
tan of 0.2 = 0.20 0.20
tan of 0.3 = 0.31 0.31
tan of 0.4 = 0.42 0.42
tan of 0.5 = 0.55 0.55
tan of 0.6 = 0.68 0.68
tan of 0.7 = 0.84 0.84
tan of 0.8 = 1.03 1.03
tan of 0.9 = 1.26 1.26
tan of 1.0 = 1.56 1.56
tan of 1.1 = 1.96 1.96
tan of 1.2 = 2.57 2.57
tan of 1.3 = 3.60 3.60
tan of 1.4 = 5.80 5.80
$
```



## Memory Allocation Functions

Table 8–1 lists and describes all the memory allocation functions found in the *Compaq C* Run-Time Library (RTL). For a more detailed description of each function, see the Reference Section.

**Table 8–1 Memory Allocation Functions**

| Function                                  | Description                                                                                                                                |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <code>brk</code> , <code>sbrk</code>      | Determine the lowest virtual address that is not used with the program.                                                                    |
| <code>calloc</code> , <code>malloc</code> | Allocate an area of memory.                                                                                                                |
| <code>cfree</code> , <code>free</code>    | Make available for reallocation the area allocated by a previous <code>calloc</code> , <code>malloc</code> , or <code>realloc</code> call. |
| <code>realloc</code>                      | Changes the size of the area pointed to by the first argument to the number of bytes given by the second argument.                         |
| <code>strdup</code>                       | Finds and points to a duplicate string.                                                                                                    |

All *Compaq C* RTL functions requiring additional storage from the heap get that storage using the *Compaq C* RTL memory allocation functions `malloc`, `calloc`, `realloc`, `free`, and `cfree`. Memory allocated by these functions is quadword-aligned.

The ANSI C standard does not include `cfree`. For this reason, it is preferable to free memory using the functionally equivalent `free` function.

The `brk` and `sbrk` functions assume that memory can be allocated contiguously from the top of your address space. However, the `malloc` function and RMS may allocate space from this same address space. You should not use the `brk` and `sbrk` functions in conjunction with RMS and *Compaq C* RTL routines that use `malloc`.

Previous versions of the VAX C RTL documentation indicated that the memory allocation routines used the *OpenVMS* RTL functions `LIB$GET_VM` and `LIB$FREE_VM` to acquire and return dynamic memory. This is no longer the case; interaction between these routines and the *Compaq C* RTL memory allocation routines is no longer problematic (although `LIB$SHOW_VM` can no longer be used to track *Compaq C* RTL `malloc` and `free` usage).

The *Compaq C* RTL memory allocation functions `calloc`, `malloc`, `realloc`, and `free` are based on the `LIB$` routines `LIB$VM_CALLOC`, `LIB$VM_MALLOC`, `LIB$VM_REALLOC` and `LIB$VM_FREE`, respectively.

The routines `VAXC$CALLOC_OPT`, `VAXC$CFREE_OPT`, `VAXC$FREE_OPT`, `VAXC$MALLOC_OPT`, and `VAXC$REALLOC_OPT` are now obsolete and should not be used in new development. However, versions of these routines that are equivalent to the standard C memory allocation routines are provided for backward compatibility.

## 8.1 Program Example

Example 8–1 shows the use of the malloc, calloc, and free functions.

### Example 8–1 Allocating and Deallocating Memory for Structures

```
/* CHAP_8_MEM_MANAGEMENT.C */
/* This example takes lines of input from the terminal until */
/* it encounters a Ctrl/Z, places the strings into an */
/* allocated buffer, copies the strings to memory allocated for */
/* structures, prints the lines back to the screen, and then */
/* deallocates all the memory used for the structures. */

#include <stdlib.h>
#include <stdio.h>
#define MAX_LINE_LENGTH 80

struct line_rec { /* Declare the structure. */
 struct line_rec *next; /* Pointer to next line. */
 char *data; /* A line from terminal. */
};

int main(void)
{
 char *buffer;

 /* Define pointers to */
 /* structure (input lines). */

 struct line_rec *first_line = NULL,
 *next_line,
 *last_line = NULL;

 /* Buffer points to memory. */
 buffer = malloc(MAX_LINE_LENGTH);

 if (buffer == NULL) { /* If error ... */
 perror("malloc");
 exit(EXIT_FAILURE);
 }

 while (gets(buffer) != NULL) { /* While not Ctrl/Z ... */
 /* Allocate for input line. */
 next_line = calloc(1, sizeof (struct line_rec));

 if (next_line == NULL) {
 perror("calloc");
 exit(EXIT_FAILURE);
 }

 /* Put line in data area. */
 next_line->data = buffer;

 if (last_line == NULL) /* Reset pointers. */
 first_line = next_line;
 else
 last_line->next = next_line;

 last_line = next_line;
 /* Allocate space for the */
 /* next input line. */
 buffer = malloc(MAX_LINE_LENGTH);
 }
}
```

(continued on next page)

### Example 8–1 (Cont.) Allocating and Deallocating Memory for Structures

```
 if (buffer == NULL) {
 perror("malloc");
 exit(EXIT_FAILURE);
 }
 }
 free(buffer); /* Last buffer always unused. */
 next_line = first_line; /* Pointer to beginning. */
 while (next_line != NULL) {
 puts(next_line->data); /* Write line to screen. */
 free(next_line->data); /* Deallocate a line. */
 last_line = next_line;
 next_line = next_line->next;
 free(last_line);
 }
 exit(EXIT_SUCCESS);
}
```

The sample input and output for Example 8–1 are as follows:

```
$ RUN EXAMPLE
line one
line two
Ctrl/Z
EXIT
line one
line two
$
```



## System Functions

The C programming language is a good choice if you wish to write operating systems. For example, much of the UNIX operating system is written in C. When writing system programs, it is sometimes necessary to retrieve or modify the environment in which the program is running. This chapter describes the *Compaq C* Run-Time Library (RTL) functions that accomplish this and other system tasks.

Table 9–1 lists and describes all the system functions found in the *Compaq C* RTL. For a more detailed description of each function, see the Reference Section.

**Table 9–1 System Functions**

| Function                                                | Description                                                                                                              |
|---------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>System Functions—Searching and Sorting Utilities</b> |                                                                                                                          |
| bsearch                                                 | Performs a binary search on an array of sorted objects for a specified object.                                           |
| qsort                                                   | Sorts an array of objects in place by implementing the quick-sort algorithm.                                             |
| <b>System Functions—Retrieving Process Information</b>  |                                                                                                                          |
| ctermid                                                 | Returns a character string giving the equivalence string of SYS\$COMMAND, which is the name of the controlling terminal. |
| cuserid                                                 | Returns a pointer to a character string containing the name of the user who initiated the current process.               |
| getcwd                                                  | Returns a pointer to the file specification for the current working directory.                                           |
| getegid, geteuid, getgid, getuid                        | Returns, in <i>OpenVMS</i> terms, group and member numbers from the user-identification code (UIC).                      |
| getenv                                                  | Searches the environment array for the current process and returns the value associated with a specified environment.    |
| getlogin                                                | Gets the login name of the user associated with the current session.                                                     |
| getpid                                                  | Returns the process ID of the current process.                                                                           |

(continued on next page)

**Table 9–1 (Cont.) System Functions**

| Function                                                                | Description                                                                                                                           |
|-------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>System Functions—Retrieving Process Information</b>                  |                                                                                                                                       |
| getppid                                                                 | Returns the parent process ID of the calling process.                                                                                 |
| getpwnam                                                                | Accesses user-name information in the user database.                                                                                  |
| getpwuid                                                                | Accesses user-ID information in the user database.                                                                                    |
| <b>System Functions—Changing Process Information</b>                    |                                                                                                                                       |
| chdir                                                                   | Changes the default directory.                                                                                                        |
| chmod                                                                   | Changes the file protection of a file.                                                                                                |
| chown                                                                   | Changes the owner user identification code (UIC) of a file.                                                                           |
| mkdir                                                                   | Creates a directory.                                                                                                                  |
| nice                                                                    | Increases or decreases the process priority to the process base priority by the amount of the argument.                               |
| putenv                                                                  | Sets an environmental variable.                                                                                                       |
| setenv                                                                  | Inserts or resets the environment variable name in the current environment list.                                                      |
| setgid, setuid                                                          | Implemented for program portability and have no functionality.                                                                        |
| sleep, usleep                                                           | Suspends the execution of the current process for at least the number of seconds indicated by its argument.                           |
| umask                                                                   | Creates a file protection mask that is used whenever a new file is created. It returns the old mask value.                            |
| <b>System Functions—Retrieving and Converting Date/Time Information</b> |                                                                                                                                       |
| asctime                                                                 | Converts a broken-down time into a 26-character string.                                                                               |
| clock                                                                   | Determines the CPU time, in microseconds, used since the beginning of the program execution.                                          |
| ctime                                                                   | Converts a time, in seconds, to an ASCII string in the form generated by the asctime function.                                        |
| decc\$fix_time                                                          | Converts <i>OpenVMS</i> binary system times to UNIX binary times.                                                                     |
| difftime                                                                | Computes the difference, in seconds, between the two times specified by its arguments.                                                |
| ftime                                                                   | Returns the elapsed time since 00:00:00, January 1, 1970, in the structure <code>timeb</code> .                                       |
| getclock                                                                | Gets the current value of the system-wide clock.                                                                                      |
| getdate                                                                 | Converts a formatted string to a time/date structure.                                                                                 |
| getitimer                                                               | Returns the value of interval timers.                                                                                                 |
| gettimeofday                                                            | Gets date and time.                                                                                                                   |
| gmtime                                                                  | Converts time units to broken-down UTC time.                                                                                          |
| localtime                                                               | Converts a time (expressed as the number of seconds elapsed since 00:00:00, January 1, 1970) into hours, minutes, seconds, and so on. |
| mktime                                                                  | Converts a local-time structure into time since the Epoch.                                                                            |

(continued on next page)

**Table 9–1 (Cont.) System Functions**

| Function                                                                | Description                                                                                                                                                                      |
|-------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>System Functions—Retrieving and Converting Date/Time Information</b> |                                                                                                                                                                                  |
| setitimer                                                               | Sets the value of interval timers.                                                                                                                                               |
| strftime, wcsftime                                                      | Places characters into an array, as controlled by a specified format string.                                                                                                     |
| strptime                                                                | Converts a character string into date and time values.                                                                                                                           |
| time                                                                    | Returns the time elapsed since 00:00:00, January 1, 1970, in seconds.                                                                                                            |
| times                                                                   | Returns the accumulated times of the current process and of its terminated child processes.                                                                                      |
| tzset                                                                   | Sets and accesses time-zone conversion.                                                                                                                                          |
| ualarm                                                                  | Sets or changes the timeout of interval timers.                                                                                                                                  |
| wcsftime                                                                | Uses date and time information stored in a <code>tm</code> structure to create a wide-character output string.                                                                   |
| <b>System Functions—Miscellaneous</b>                                   |                                                                                                                                                                                  |
| VAXC\$CRTL_INIT                                                         | Initializes the run-time environment and establishes an exit and condition handler, which makes it possible for <i>Compaq C</i> RTL functions to be called from other languages. |

Example 9–1 shows how the `cuserid` function is used.

#### Example 9–1 Accessing the User Name

```

/* CHAP_9_GET_USER.C */
/* Using cuserid, this program returns the user name. */
#include <stdio.h>
main()
{
 static char string[L_cuserid];
 cuserid(string);
 printf("Initiating user: %s\n", string);
}

```

If a user named TOLLIVER runs the program, the following is displayed on `stdout`:

```

$ RUN EXAMPLE1
Initiating user: TOLLIVER

```

Example 9–2 shows how the `getenv` function is used.

### Example 9–2 Accessing Terminal Information

```
/* CHAP_9_GETTERM.C */
/* Using getenv, this program returns the terminal. */
#include <stdio.h>
#include <stdlib.h>

main()
{
 printf("Terminal type: %s\n", getenv("TERM"));
}
```

Running Example 9–2 on a VT100 terminal in 132-column mode displays the following:

```
$ RUN EXAMPLE3
Terminal type: vt100-132
```

Example 9–3 shows how to use `getenv` to find the user’s default login directory and how to use `chdir` to change to that directory.

### Example 9–3 Manipulating the Default Directory

```
/* CHAP_9_CHANGE_DIR.C */
/* This program performs the equivalent of the DCL command */
/* SET DEFAULT SYS$LOGIN. However, once the program exits, the */
/* directory is reset to the directory from which the program */
/* was run. */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
 char *dir;
 int i;

 dir = getenv("HOME");
 if ((i = chdir(dir)) != 0) {
 perror("Cannot set directory");
 exit(0);
 }

 printf("Current directory: %s\n", dir);
}
```

Running Example 9–3 displays the following:

```
$ RUN EXAMPLE4
Current directory: dba0:[tolliver]
$
```

Example 9–4 shows how to use the `time`, `localtime`, and `strftime` functions to print the correct date and time at the terminal.



#### Example 9–4 Printing the Date and Time

```
/* CHAP_9_DATE_TIME.C */
/* The time function returns the time in seconds; the localtime */
/* function converts the time to hours, minutes, and so on; */
/* the strftime function uses these values to obtain a string */
/* in the desired format. */

#include <time.h>
#include <stdio.h>

#define MAX_STRING 80

main()
{
 struct tm *time_structure;
 time_t time_val;
 char output_str[MAX_STRING];

 time(&time_val);
 time_structure = localtime(&time_val);

 /* Print the date */
 strftime(output_str, MAX_STRING,
 "Today is %A, %B %d, %Y", time_structure);

 printf("%s\n", output_str);

 /* Print the time using a 12-hour clock format. */
 strftime(output_str, MAX_STRING,
 "The time is %I:%M %p", time_structure);

 printf("%s\n", output_str);
}
```

Running Example 9–4 displays the following:

```
$ RUN EXAMPLE5
Today is Thursday, May 20, 1993
The time is 10:18 AM
$
```



---

## Developing International Software

This chapter describes typical features of international software and the features provided with the *Compaq C* Run-Time Library (RTL) that enable you to design and implement international software.

See the Reference Section for more detailed information on the functions described in this chapter.

### 10.1 Internationalization Support

The *Compaq C* RTL has added capabilities to allow application developers to create international software. The *Compaq C* RTL obtains information about a language and a culture by reading this information from *locale* files.

#### 10.1.1 Installation

If you are using these *Compaq C* RTL capabilities, you must install a separate kit to provide these files to your system.

The save set, VMSI18N0nn, is provided on the same media as the *OpenVMS* operating system.

To install this save set, follow the standard *OpenVMS* installation procedures using this save-set name as the name of the kit. There are several categories of locales that you can select to install. You can select as many locales as you need by answering the following prompts:

```
Do you want European and US support?
Do you want Chinese support?
Do you want Japanese support?
Do you want Korean support?
Do you want Thai support?
Do you want the Unicode converters?
```

This kit also has an Installation Verification Procedure that Compaq recommends you run to verify the correct installation of the kit.

#### 10.1.2 Unicode Support

In *OpenVMS* Version 7.2, the *Compaq C* Run-Time Library added the Universal Unicode locale, which is distributed with the *OpenVMS* system, not with the VMSI18N0nn kit. The name of the Unicode locale is:

```
UTF8-20
```

Like those locales shipped with the VMSI18N0nn kit, the Unicode locale is located at the standard location referred to by the SYS\$I18N\_LOCALE logical name.

The UTF8-20 Unicode is based on Unicode standard Version V2.0. The Unicode locale uses UCS-4 as wide-character encoding and UTF-8 as multibyte character encodings.

*Compaq C* RTL also includes converters that perform conversions between Unicode and any other supported character sets. The expanded set of converters includes converters for UCS-2, UCS-4, and UTF-8 Unicode encoding. The Unicode converters can be used by the ICONV CONVERT utility and by the iconv family of functions in the *Compaq C* Run-Time Library.

In *OpenVMS* Version 7.2, the *Compaq C* Run-Time Library added Unicode character set converters for Microsoft Code Page 437.

## 10.2 Features of International Software

International software is software that can support multiple languages and cultures. An international program should be able to:

- Display messages in the user's own language. This includes screen displays, error messages, and prompts.
- Handle culture-specific information such as:
  - Date and time formatting

The conventions for representing dates and times vary from one country to another. For example, in the US the month is given first; in the UK the day is specified first. Therefore, the date 12/5/1993 is interpreted as December 5, 1993 in the US, and as May 12, 1993 in the UK.
  - Numeric formatting

The character that represents the decimal point (the radix character) and the thousands separator character vary from one country to another. For example, in the UK the period (.) is used to represent the radix character, and the comma is used as a separator. However, in Germany, the comma is used as the radix character and the period is the separator character. Therefore, the number 2,345.67 in the UK is the same as 2.345,67 in Germany.
  - Monetary formatting

Currency values are represented by different symbols and can be formatted using a variety of separator characters, depending on the currency.
- Handle different coded character sets (not just ASCII).
- Handle a mixture of single and multibyte characters.
- Provide multipass string comparisons.

String comparison functions such as `strcmp` compare strings by comparing the codepoint values of the characters in the strings. However, some languages require more complex comparisons to correctly sort strings.

To meet the previous requirements, an application should not make any assumptions about the language, local customs, or the coded character set used. All this localization data should be defined separately from the program, and only bound to it at run time.

The rest of this chapter describes how you can create international software using *Compaq C*.

## 10.3 Developing International Software Using *Compaq C*

The *Compaq C* environment provides the following facilities to create international software:

- A method for separating localization data from a program.  
Localization data is held in a database known as a *locale*. This stores all the language and culture information required by a program. See Section 10.4 for details of the structure of locales.  
A program specifies what locales to use by calling the `setlocale` function. See Section 10.5 for more information.
- A method of separating message text from the program source.  
This is achieved using *message catalogs* that store all the messages for an application. The message catalog is linked to the application at run time. This means that the messages can be translated into different languages and then the required language version is selected at run time. See Section 10.6.
- *Compaq C* RTL functions that are sensitive to localization data.  
The *Compaq C* RTL includes functions for:
  - Converting between different codesets. See Section 10.7.
  - Handling culture-specific information. See Section 10.8.
  - Multipass string collation. See Section 10.10.
- A special wide-character data type defined in the *Compaq C* RTL makes it easier to handle codesets that have a mixture of single and multibyte characters. A set of functions is also defined to support this wide-character data type. See Section 10.9.

## 10.4 Locales

A locale consists of different categories, each of which determines one aspect of the international environment. Table 10–1 lists the categories in a locale and describes the information in each.

**Table 10–1 Locale Categories**

| Category    | Description                                                          |
|-------------|----------------------------------------------------------------------|
| LC_COLLATE  | Contains information about collating sequences.                      |
| LC_CTYPE    | Contains information about character classification.                 |
| LC_MESSAGES | Defines the answers that are expected in response to yes/no prompts. |
| LC_MONETARY | Contains monetary formatting information.                            |
| LC_NUMERIC  | Contains information about formatting numbers.                       |
| LC_TIME     | Contains time and date information.                                  |

The locales provided by *Compaq* reside in the directory defined by the `SYS$I18N_LOCALE` logical name. The file-naming convention for locales is:

`language_country_codeset.locale`

Where:

- *language* is the mnemonic for the language. For example, EN indicates an English locale.
- *country* is the mnemonic for the country. For example, GB indicates a British locale.
- *codeset* is the name of the ISO standard codeset for the locale. For example, ISO8859-1 is the ISO 8859 codeset for the Western European languages. See Section 10.7 for more information about the codesets supported.

## 10.5 Using the `setlocale` Function to Set Up an International Environment

An application sets up its international environment at run time by calling the `setlocale` function. The international environment is set up in one of two ways:

- The environment is defined by one locale. In this case, each of the locale categories is defined by the same locale.
- Categories are defined separately. This lets you define a mixed environment that uses different locales depending on the operation performed. For example, if an English user has some Spanish files that are to be processed by an application, the `LC_COLLATE` category could be defined by a Spanish locale while the other categories are defined by an English locale. To do this you would call `setlocale` once for each category.

The syntax for the `setlocale` function is:

```
char *setlocale(int category, const char *locale)
```

Where:

- *category* is either the name of a category, or `LC_ALL`. Specifying `LC_ALL` means that all the categories are defined by the same locale. Specify a category name to set up a mixed environment.
- *locale* is one of the following:

- The name of the locale to use.

If you want users to specify the locale interactively, your application could prompt the user for a locale name, and then pass the name as an argument to the `setlocale` function. A locale name has the following format:

```
language_country.codeset[@modifier]
```

For example, `setlocale(LC_COLLATE, "en_US.ISO8859-1")` selects the locale `en_US.ISO8859-1` for the `LC_COLLATE` category.

- ""

This causes the function to use logical names to determine the locale for the category specified. See [Specifying the Locale Using Logical Names](#) for details.

If an application does not call the `setlocale` function, the default locale is the C locale. This allows such applications to call those functions that use information in the current locale.

### Specifying the Locale Using Logical Names

If the `setlocale` function is called with "" as the *locale* argument, the function checks for a number of logical names to determine the locale name for the category specified.

There are a number of logical names that users can set up to define their international environment:

- Logical name corresponding to a category  
For example, the `LC_NUMERIC` logical name defines the locale associated with the `LC_NUMERIC` category within the user's environment.
- `LC_ALL`
- `LANG`  
The `LANG` logical name defines the user's language.

In addition to the logical names defined by a user, there are a number of systemwide logical names, set up during system startup, that define the default international environment for all users on a system:

- `SYS$category`  
Where *category* is the name of a category. This specifies the system default for that category.
- `SYS$LC_ALL`
- `SYS$LANG`

The `setlocale` function checks for user-defined logical names first, and if these are not defined, it checks the system logical names.

## 10.6 Using Message Catalogs

An important requirement for international software is that it should be able to communicate with the user in the user's own language. The messaging system enables program messages to be created separately from the program source, and linked to the program at run time.

Messages are defined in a message text source file, and compiled into a message catalog using the `GENCAT` command. The message catalog is accessed by a program using the functions provided in the *Compaq C* RTL.

The functions provided to access the messages in a catalog are:

- The `catopen` function, which opens a specified catalog ready for use.
- The `catgets` function, which enables the program to read a specific message from a catalog.
- The `catclose` function, which closes a specified catalog. Open message catalogs are also closed by the `exit` function.

For information on generating message catalogs, see the `GENCAT` command description in the *OpenVMS* system documentation.

## 10.7 Handling Different Character Sets

The *Compaq C* RTL supports a number of state-independent codesets and codeset encoding schemes that contain the ASCII encoded Portable Character Set. It does not support state-dependent codesets. The codesets supported are:

- ISO8859-*n*  
where *n* = 1,2,5,7,8 or 9. This covers codesets for North America, Europe (West and East), Israel, and Turkey.
- eucJP, SJIS, DECKANJI, SDECKANJI: Codesets used in Japan.
- eucTW, DECHANYU, BIG5, DECHANZI: Chinese codesets used in China (PRC), Hong-Kong, and Taiwan.
- DECKOREAN: Codeset used in Korea.

### 10.7.1 Charmap File

The characters in a codeset are defined in a charmap file. The charmap files supplied by Compaq are located in the directory defined by the SYS\$I18N\_LOCALE logical name. The file type for a charmap file is .CMAP.

### 10.7.2 Converter Functions

As well as supporting different coded character sets, the *Compaq C* RTL provides the following converter functions that enable you to convert characters from one codeset to another:

- `iconv_open`—Specifies the type of conversion. It allocates a conversion descriptor required by the `iconv` function.
- `iconv`—Converts characters in a file to the equivalent characters in a different codeset. The converted characters are stored in a separate file.
- `iconv_close`—Deallocates a conversion descriptor and the resources allocated to the descriptor.

### 10.7.3 Using Codeset Converter Files

The file-naming convention for codeset converters is:

*fromcode\_tocode.iconv*

Where *fromcode* is the name of the source codeset, and *tocode* is the name of the codeset to which characters are converted.

You can add codeset converters to a given system by installing the converter files in the directory pointed by the logical name SYS\$I18N\_ICONV.

Codeset converter files can be implemented either as table-based conversion files or as algorithm-based converter files created as *OpenVMS* shareable images.

#### Creating a Table-based Conversion File

The following summarizes the necessary steps to create a table-based codeset converter file:

1. Create a text file that describes the mapping between any character from the source codeset to the target codeset. For the format of this file, see the DCL command `ICONV COMPILE` in the *OpenVMS New Features Manual*, which processes such a file and creates a codeset converter table file.



2. Copy the resulting file from the previous step to the directory pointed by the logical SYS\$I18N\_ICONV, assuming you have the privilege to do so.

### Creating an Algorithm-based Conversion File

To create an algorithm-based codeset converter file implemented as a shareable image, follow these steps:

1. Create C source files that implement the codeset converter. The API is documented in the public header file <iconv.h> as follows:
  - The universal entry point `_u_iconv_open` is called by the *Compaq C* RTL routine `iconv_open` to initialize a conversion.
  - `_u_iconv_open` returns to `iconv_open` a pointer to the structure `__iconv_extern_obj_t`.
  - Within this structure, the converter exports its own conversion entry point and conversion close routine, which are called by the *Compaq C* RTL routines `iconv` and `iconv_close`, respectively.
  - The major and minor identifier fields are required by `iconv_open` to test for a possible mismatch between the library and the converter. The converter usually assigns the constants `__ICONV_MAJOR` and `__ICONV_MINOR`, defined in the <iconv.h> header file.
  - The field `tcs_mb_cur_max` is used only by the DCL command `ICONV CONVERT` to optimize its buffer usage. This field reflects the maximum number of bytes that comprise a single character in the target codeset, including the shift sequence (if any).
2. Compile and link the modules that comprise the codeset converter as an *OpenVMS* shareable image, making sure that the file name adheres to the preceding conventions.
3. Copy the resulting file from the previous step to the directory pointed by the logical SYS\$I18N\_ICONV, assuming you have the privilege to do so.

### Some Final Notes

By default, SYS\$I18N\_ICONV is a search list where the first directory in the list SYS\$SYSROOT:[SYS\$I18N.ICONV.USER] is meant for use as a site-specific repository for `iconv` codeset converters.

The number of codesets and locales installed vary from system to system. Check the SYS\$I18N directory tree for the codesets, converters, and locales installed on your system.

## 10.8 Handling Culture-Specific Information

Each locale contains the following cultural information:

- Date and time information  
The `LC_TIME` category defines the conventions for writing date and time, the names of the days of the week, and the names of months of the year.
- Numeric information  
The `LC_NUMERIC` category defines the conventions for formatting nonmonetary values.
- Monetary information

The `LC_MONETARY` category defines currency symbols and the conventions used to format monetary values.

- Yes and no responses

The `LC_MESSAGES` category defines the strings expected in response to yes/no questions.

You can extract some of this cultural information using the `nl_langinfo` function and the `localeconv` function. See Section 10.8.1.

### 10.8.1 Extracting Cultural Information From a Locale

The `nl_langinfo` function returns a pointer to a string that contains an item of information obtained from the program's current locale. The information you can extract from the locale is:

- Date and time formats
- The names of the days of the week, and months of the year in the local language
- The radix character
- The character used to separate groups of digits in nonmonetary values
- The currency symbol
- The name of the codeset for the locale
- The strings defined for responses to yes/no questions

The `localeconv` function returns a pointer to a data structure that contains numeric formatting and monetary formatting data from the `LC_NUMERIC` and `LC_MONETARY` categories.

### 10.8.2 Date and Time Formatting Functions

The functions that use the date and time information are:

- `strftime`—Takes date and time values stored in a data structure and formats them into an output string. The format of the output string is controlled by a format string.
- `strptime`—Converts a string (of type `char`) into date and time values. A format string defines how the string is interpreted.
- `wcsftime`—Does the same as `strftime` except that it creates a wide-character string.

### 10.8.3 Monetary Formatting Function

The `strfmon` function uses the monetary information in a locale to convert a number of values into a string. The format of the string is controlled by a format string.

### 10.8.4 Numeric Formatting

The information in `LC_NUMERIC` is used by various functions. For example, `strtod`, `wcstod`, and the `print` and `scan` functions determine the radix character from the `LC_NUMERIC` category.

## 10.9 Functions for Handling Wide Characters

A character can be represented by single-byte or multibyte values depending on the codeset. To make it easier to handle both single-byte and multibyte characters in the same way, the *Compaq C* RTL defines a wide-character data type, `wchar_t`. This data type can store characters that are represented by 1-, 2-, 3-, or 4-byte values.

The functions provided to support wide characters are:

- Character classification functions. See Section 10.9.1.
- Case conversion functions. See Section 10.9.2.
- Input and output functions. See Section 10.9.3.
- Multibyte to wide-character conversion functions. See Section 10.9.4.
- Wide-character to multibyte conversion functions. See Section 10.9.4.
- Wide-character string manipulation functions. See Section 10.9.5.
- Wide-character string collation and comparison functions. See Section 10.10.

### 10.9.1 Character Classification Functions

The `LC_CTYPE` category in a locale classifies the characters in the locale's codeset into different types (alphabetic, numeric, lowercase, uppercase, and so on). There are two sets of functions, one for wide characters and one for single-byte characters, that test whether a character is of a specific type. The `is*` functions test single-byte characters, and the `isw*` functions test wide characters.

For example, the `iswalnum` function tests if a wide character is classed as either alphabetic or numeric. It returns a nonzero value if the character is one of these types. For more information about the classification functions, see Chapter 3 and the Reference Section.

### 10.9.2 Case Conversion Functions

The `LC_CTYPE` category defines mapping between pairs of characters of the locale. The most common character mapping is between uppercase and lowercase characters. However, a locale can support more than just case mappings.

Two functions are provided to map one character to another according to the information in the `LC_CTYPE` category of the locale:

- `wctrans`—Looks for the named mapping (predefined in the locale) between characters.
- `towctrans`—Maps one character to another according to the named mapping given to the `wctrans` function.

Two functions are provided for character case mapping:

- `towlower`—Maps an uppercase wide character to its lowercase equivalent.
- `toupper`—Maps a lowercase wide character to its uppercase equivalent.

For more information about these functions, see the Reference Section.

### 10.9.3 Functions for Input and Output of Wide Characters

The set of input and output functions manages wide characters and wide-character strings.

#### Read Functions

The functions for reading wide characters and wide-character strings are `fgetwc`, `fgetws`, `getwc`, and `getwchar`.

There is also an `ungetwc` function that pushes a wide character back into the input stream.

#### Write Functions

The functions for writing wide characters and wide-character strings are `fputwc`, `fputws`, `putwc`, and `putwchar`.

#### Scan Functions

All the scan functions allow for a culture-specific radix character, as defined in the `LC_NUMERIC` category of the current locale.

The `%lc`, `%C`, `%ls`, and `%S` conversion specifiers enable the scan functions `fwscanf`, `wscanf`, `swscanf`, `fscanf`, `scanf`, and `sscanf` to read in wide characters.

#### Print Functions

All the print functions can format numeric values according to the data in the `LC_NUMERIC` category of the current locale.

The `%lc`, `%C` and `%ls`, `%S` conversion specifiers used with print functions convert wide characters to multibyte characters and print the resulting characters.

See Chapter 2 for details of all input and output functions.

### 10.9.4 Functions for Converting Multibyte and Wide Characters

Wide characters are used internally by an application to manage single-byte or multibyte characters. However, text files are generally stored in multibyte character format. To process these files, the multibyte characters need converting to wide-character format. This can be achieved using the following functions:

- `mbtowc`, `mbrtowc`, `btowc`—Convert one multibyte character to a wide character.
- `mbsrtowcs`, `mbstowcs`—Convert a multibyte character string to a wide-character string.

Similarly, the following functions convert wide characters into their multibyte equivalent:

- `wctomb`, `wctomb`, `wctob`—Convert a single wide character to a multibyte character.
- `wcsrtombs`, `wcstombs`—Convert a wide-character string to a multibyte character string.

Associated with these conversion functions, the `mblen` and `mbrlen` functions are used to determine the size of a multibyte character.

Several of the wide-character functions take an argument of type "pointer to `mbstate_t`", where `mbstate_t` is an opaque datatype (like `FILE` or `fpos_t`) intended to keep the conversion state for the state-dependent codesets.

### 10.9.5 Functions for Manipulating Wide-Character Strings and Arrays

The *Compaq C* RTL contains a set of functions (the `wcs*` and `wmem*` functions) that manipulate wide-character strings. For example, the `wscat` function appends a wide-character string to the end of another string in the same way that the `strcat` function works on character strings of type *char*.

See Chapter 3 for details of the string manipulation functions.

## 10.10 Collating Functions

In an international environment, string comparison functions need to allow for multipass collations. The collation requirements include:

- Ordering accented characters.
- Collating a character sequence as a single character. For example, *ch* in Spanish should be collated after *c* but before *d*.
- Collating a single character as a two-character sequence.
- Ignoring some characters.

Collating information is stored in the `LC_COLLATE` category of a locale. The *Compaq C* RTL includes the `strcoll` and `wscoll` functions that use this collating information to compare two strings.

Multipass collations by `strcoll` or `wscoll` can be slower than using the `strcmp` or `wscmp` functions. If your program needs to do many string comparisons using `strcoll` or `wscoll`, it may be quicker to transform the strings once, using the `strxfrm` or `wcsxfrm` function, and then use the `strcmp` or `wscmp` function.

The term *collation* refers to the relative order of characters. The collation order is locale-specific and might ignore some characters. For example, an American dictionary ignores the hyphen in words and lists *take-out* between *takeoff* and *takeover*.

*Comparison*, on the other hand, refers to the examination of characters for sameness or difference. For example, *takeout* and *take-out* are different words, although they may collate the same.

Suppose an application sorts a list of words so it can later perform a binary search on the list to quickly retrieve a word. Using `strcmp`, *take-in*, *take-out*, and *take-up* would be grouped in one part of the table. Using `strcoll` and a locale that ignores hyphens, *take-out* would be grouped with *takeoff* and *takeover*, and would be considered a duplicate of *takeout*. To avoid a binary search finding *takeout* as a duplicate of *take-out*, an application would most likely use `strcmp` rather than `strcoll` for forming a binary tree.



## Date/Time Functions

This chapter describes the date/time functions available with *Compaq C* for *OpenVMS* Systems. For more detailed information on each function, see the Reference Section.

**Table 11–1 Date/Time Functions**

| Function                  | Description                                                                                                                                 |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <code>asctime</code>      | Converts a broken-down time from <code>localtime</code> into a 26-character string.                                                         |
| <code>ctime</code>        | Converts a time, in seconds, since 00:00:00, January 1, 1970 to an ASCII string of the form generated by the <code>asctime</code> function. |
| <code>ftime</code>        | Returns the elapsed time since 00:00:00, January 1, 1970 in the structure pointed to by its argument.                                       |
| <code>getclock</code>     | Gets the current value of the systemwide clock.                                                                                             |
| <code>gettimeofday</code> | Gets the date and time.                                                                                                                     |
| <code>gmtime</code>       | Converts time units to GMT (Greenwich Mean Time).                                                                                           |
| <code>localtime</code>    | Converts a time (expressed as the number of seconds elapsed since 00:00:00, January 1, 1970) into hours, minutes, seconds, and so on.       |
| <code>mktime</code>       | Converts a local time structure to a calendar time value.                                                                                   |
| <code>time</code>         | Returns the time elapsed since 00:00:00, January 1, 1970, in seconds.                                                                       |
| <code>tzset</code>        | Sets and accesses time-zone conversion.                                                                                                     |

Also, the time-related information returned by `fstat` and `stat` uses the new date/time model described in Section 11.1.

### 11.1 Date/Time Support Models

Beginning with *OpenVMS* Version 7.0, the *Compaq C* RTL changed its date/time support model from one based on local time to one based on Universal Coordinated Time (UTC). This allows the *Compaq C* RTL to implement ANSI C/POSIX functionality that previously could not be implemented. A UTC time-based model also makes the *Compaq C* RTL compatible with the behavior of the Tru64 UNIX time functions.

By default, newly compiled programs will generate entry points into UTC-based date/time routines.

For compatibility with *OpenVMS* systems prior to Version 7.0, previously compiled programs that relink on an *OpenVMS* Version 7.0 system will retain local-time-based date/time support. Relinking alone will not access UTC support.

Compiling programs with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined will also enable local-time-based entry points. That is, the new *OpenVMS* Version 7.0 date/time functions will not be enabled.

Functions with both UTC-based and local-time-based entry points are:

|           |          |
|-----------|----------|
| ctime     | mktime   |
| fstat     | stat     |
| ftime     | strftime |
| gmtime    | time     |
| localtime | wcsftime |

---

**Note**

---

Introducing a UTC-based, date/time model implies a certain loss of performance because time-related functions supporting UTC must read and interpret time-zone files instead of doing simple computations in memory as was done for the date/time model based on local time.

To decrease this performance degradation, *OpenVMS* Version 7.1 and higher can maintain the processwide cache of time-zone files. The size of the cache (that is, the number of files in the memory) is determined by the value of the DECC\$TZ\_CACHE\_SIZE logical name. The default value is 2.

Because the time-zone files are relatively small (about 3 blocks each), consider defining DECC\$TZ\_CACHE\_SIZE as the maximum number of time zones used by the application. For example, the default cache size fits an application that does not switch time zones during the run and runs on a system where the TZ environment variable is defined with both Standard and Summer time zone.

---

## 11.2 Overview of Date/Time Functions

In the UTC-based model, times are represented as seconds since the Epoch. The Epoch is defined as the time 0 hours, 0 minutes, 0 seconds, January 1, 1970 UTC. Seconds since the Epoch is a value interpreted as the number of seconds between a specified time and the Epoch.

The functions `time` and `ftime` return the time as seconds since the Epoch.

The functions `ctime`, `gmtime`, and `localtime` take as their argument a time value that represents the time in seconds from the Epoch.

The function `mktime` converts a broken-down time, expressed as local time, into a time value in terms of seconds since the Epoch.

The values `st_ctime`, `st_atime`, and `st_mtime` returned in the `stat` structure by the `stat` and `fstat` functions are also in terms of UTC.

Time support new to *OpenVMS* Version 7.0 includes the functions `tzset`, `gettimeofday`, and `getclock`, and the external variables `tzname`, `timezone`, and `daylight`.

The UTC-based time model enables the *Compaq C* RTL to:

- Implement the ANSI C `gmtime` function, which returns a structure in terms of GMT time.
- Specify the ANSI `tm_isdst` field of the `tm` structure, which specifies whether daylight savings time is in effect.



- Provide time-related POSIX and X/Open extensions (such as the `tzset` function (which lets you get time information from any time zone), and the external variables `tzname`, `timezone`, and `daylight`).
- Correctly compute the local time for times in the past, something that the time functions like `localtime` need to do.
- Enable `localtime` and `gmtime`, through the use of feature-test macros (see Section 1.5), to return two additional fields: `tm_zone` (an abbreviation of the time-zone name) and `tm_gmtoff` (the offset from UTC in seconds) in the `tm` structure they return.

### 11.3 *Compaq C* RTL Date/Time Computations—UTC and Local Time

Universal Coordinated Time (UTC) is an international standard for measuring time of day. Under the UTC time standard, zero hour occurs when the Greenwich Meridian is at midnight. UTC has the advantage of always increasing, unlike local time, which can go backwards/forwards depending on daylight saving time.

Also, UTC has two additional components:

- A measure of inaccuracy (optional)
- A time-differential factor, which is an offset applied to UTC to derive local time.

The time-differential factor associates each local time zone with UTC; the time differential factor is applied to UTC to derive local time. (Local times can vary up to –12 hours West of the Greenwich Meridian and +13 hours East of it).

For the *Compaq C* RTL time support to work correctly on *OpenVMS* Version 7.0 and higher, the following must be in place:

- Your *OpenVMS* system must be correctly configured to use a valid *OpenVMS* TDF. Make sure this is set correctly by checking the value of the `SYS$TIMEZONE_DIFFERENTIAL` logical. This logical should contain the time difference added to UTC to arrive at your local time.
- Your *OpenVMS* installation must correctly set the local time zone that describes the location that you want to be your default local time zone. In general, this is the local time zone in which your system is running.

For more information, see the section on setting up your system to compensate for different time zones in your *OpenVMS System Manager's Manual: Essentials*.

The *Compaq C* RTL uses local time-zone conversion rules to compute local time from UTC, as follows:

1. The *Compaq C* RTL internally computes time in terms of UTC.
2. The *Compaq C* RTL then uses time-zone conversion rules to compute a time-differential factor to apply to UTC to derive local time. See the `tzset` function in the Reference Section of this manual for more information on the time-zone conversion rules.

By default, the time-zone conversion rules used for computing local time from UTC are specified in time-zone files defined by the SYS\$LOCALTIME and SYS\$POSIXRULES system logicals. These logicals are set during an *OpenVMS* installation to point to time-zone files that represent the system's best approximation to local wall-clock time:

- SYS\$LOCALTIME defines the time-zone file containing the default conversion rules used by the *Compaq C* RTL to compute local time.
- SYS\$POSIXRULES defines the time-zone file that specifies the default rules to be applied to POSIX style time zones that do not specify when to change to summer time and back.

SYS\$POSIXRULES can be the same as SYS\$LOCALTIME. See the tzset function for more information.

## 11.4 Time-Zone Conversion Rule Files

The time-zone files pointed to by the SYS\$LOCALTIME and SYS\$POSIXRULES logicals are part of a public-domain, time-zone support package installed on *OpenVMS* Version 7.0 and higher systems.

This support package includes a series of source files that describe the time-zone conversion rules for computing local time from UTC in worldwide time zones. *OpenVMS* Version 7.0 and higher systems provide a time-zone compiler called ZIC. The ZIC compiler compiles time-zone source files into binary files that the *Compaq C* RTL reads to acquire time-zone conversion specifications. For more information on the format of these source files, see the *OpenVMS* system documentation for ZIC.

The time-zone files are organized as follows:

- The root time-zone directory is SYS\$COMMON:[SYS\$TIMEZONE.SYSTEM]. The system logical SYS\$TZDIR is set during installation to point to this area.
- Time-zone source files are found in SYS\$COMMON:[SYS\$TIMEZONE.SYSTEM.SOURCES].
- Binary time-zone files use SYS\$COMMON:[SYS\$TIMEZONE.SYSTEM] as their root directory. Some binaries reside in this directory while others reside in its subdirectories.
- Binaries residing in subdirectories are time-zone files that represent specific time zones in a larger geographic area. For example, SYS\$COMMON:[SYS\$TIMEZONE.SYSTEM] contains a subdirectory for the United States and a subdirectory for Canada, because each of these geographic locations contains several time zones. Each time zone in the US is represented by a time-zone file in the United States subdirectory. Each time zone in Canada is represented by a time-zone file in the Canada subdirectory.

Several of the time-zone files have names based on acronyms for the areas that they represent. Table 11–2 lists these acronyms.

**Table 11–2 Time-zone Filename Acronyms**

| Time-Zone<br>Acronym | Description                           |
|----------------------|---------------------------------------|
| CET                  | Central European Time                 |
| EET                  | Eastern European Time                 |
| Factory              | Specifies No Time Zone                |
| GB-Eire              | Great Britain/Ireland                 |
| GMT                  | Greenwich Mean Time                   |
| NZ                   | New Zealand                           |
| NZ-CHAT              | New Zealand, Chatham Islands          |
| MET                  | Middle European Time                  |
| PRC                  | Peoples Republic of China             |
| ROC                  | Republic of China                     |
| ROK                  | Republic of Korea                     |
| SystemV              | Specific to System V operating system |
| UCT                  | Universal Coordinated Time            |
| US                   | United States                         |
| UTC                  | Universal Coordinated Time            |
| Universal            | Universal Coordinated Time            |
| W-SU                 | Middle European Time                  |
| WET                  | Western European Time                 |

A mechanism is available for you to define and implement your own time-zone rules. For more information, see the *OpenVMS* system documentation on the ZIC compiler and the description of tzset in the reference section of this manual.

Also, the SYS\$LOCALTIME and SYS\$POSIXRULES system logicals can be redefined to user-supplied time zones.

## 11.5 Sample Date/Time Scenario

The following example and explanation shows how to use the *Compaq C* RTL time functions to print the current time:

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

main ()
{
 time_t t;

 t = time((time_t)0);
 printf ("The current time is: %s\n",asctime (localtime (&t)));
}
```

This example:

1. Calls the time function to get the current time in seconds since the Epoch, in terms of UTC.

2. Passes this value to the `localtime` function, which uses time-conversion information as specified by `tzset` to determine which time-zone conversion rules should be used to compute local time from UTC. By default, these rules are specified in the file defined by `SYS$LOCALTIME`:
  - a. For a user in the Eastern United States interested in their local time, `SYS$LOCALTIME` would be defined during installation to `SYS$COMMON:[SYS$ZONEINFO.US]EASTERN`, the time-zone file containing conversion rules for the Eastern U.S. time zone..
  - b. If the local time falls during daylight savings time (DST), `SYS$COMMON:[SYS$ZONEINFO.US]EASTERN` indicates that a time differential factor of  $-4$  hours needs to be applied to UTC to get local time.  
  
If the local time falls during Eastern standard time (EST), `SYS$COMMON:[SYS$ZONEINFO.US]EASTERN` indicates that a time differential factor of  $-5$  hours needs to be applied to UTC to get local time.
  - c. The *Compaq C* RTL applies  $-4$  (DST) or  $-5$  (EST) to UTC, and `localtime` returns the local time in terms of a `tm` structure.
3. Pass this `tm` structure to the `asctime` function to print the local time in a readable format.

# Reference Section

---

This section alphabetically describes the functions contained in the *Compaq C* Run-Time Library (RTL).



---

**abort**

Sends the signal SIGABRT that terminates execution of the program.

**Format**

```
#include <stdlib.h>
void abort (void);
```

## abs

---

## abs

Returns the absolute value of an integer.

### Format

```
#include <stdlib.h>

int abs (int x);
```

### Argument

**x**  
An integer.

### Return Value

|          |                                                                                                                                                                                                                |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>x</b> | The absolute value of the input argument. If the argument is <code>LONG_MIN</code> , <code>abs</code> returns <code>LONG_MIN</code> because <code>-LONG_MIN</code> cannot fit in an <code>int</code> variable. |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



---

access

Checks a file to see whether a specified access mode is allowed. This function only checks UIC protection; ACLs are not checked.

---

Note

The access function does not accept network files as arguments.

---

Format

```
#include <unistd.h>

int access (const char *file_spec, int mode);
```

Arguments

**file\_spec**  
A character string that gives an OpenVMS or UNIX style file specification. The usual defaults and logical name translations are applied to the file specification.

**mode**  
Interpreted as shown in Table REF–1.

Table REF–1 Interpretation of the mode Argument

| Mode Argument | Access Mode                     |
|---------------|---------------------------------|
| F_OK          | Tests to see if the file exists |
| X_OK          | Execute                         |
| W_OK          | Write (implies delete access)   |
| R_OK          | Read                            |

Combinations of access modes are indicated by ORing the values. For example, to check to see if a file has RWED access mode, invoke access as follows:

```
access (file_spec, R_OK | W_OK | X_OK);
```

Return Values

- 0 Indicates that the access is allowed.
- 1 Indicates that the access is not allowed.

Example

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

## **access**

```
main()
{
 if (access("sys$login:login.com", F_OK)) {
 perror("ACCESS - FAILED");
 exit(2);
 }
}
```

---

## acos

Returns the arc cosine of its argument.

### Format

```
#include <math.h>

double acos (double x);
float acosf (float x); (Alpha only)
long double acosl (long double x); (Alpha only)
double acosd (double x); (Alpha only)
float acosdf (float x); (Alpha only)
long double acosdl (long double x); (Alpha only)
```

### Argument

**x**

A radian expressed as a real value in the domain  $[-1,1]$ .

### Description

The `acos` functions compute the principal value of the arc cosine of  $x$  in the range  $[0,\pi]$  radians for  $x$  in the domain  $[-1,1]$ .

The `acosd` functions compute the principal value of the arc cosine of  $x$  in the range  $[0,180]$  degrees for  $x$  in the domain  $[-1,1]$ .

For  $\text{abs}(x) > 1$ , the value of `acos(x)` is 0, and `errno` is set to `EDOM`.

## **acosh** *(Alpha only)*

---

## **acosh** *(Alpha only)*

Returns the hyperbolic arc cosine of its argument.

### **Format**

```
#include <math.h>

double acosh (double x);
float acoshf (float x);
long double acoshl (long double x);
```

### **Argument**

**x**

A radian expressed as a real value in the domain [1, +Infinity].

### **Description**

The acosh functions return the hyperbolic arc cosine of  $x$  for  $x$  in the domain [1, +Infinity], where  $\text{acosh}(x) = \ln(x + \sqrt{x^2 - 1})$ .

The acosh function is the inverse function of cosh where  $\text{acosh}(\cosh(x)) = |x|$ .  
 $x < 1$  is an invalid argument.

---

**[w]addch**

Add a character to the window at the current position of the cursor.

**Format**

```
#include <curses.h>
int addch (char ch);
int waddch (WINDOW *win, char ch);
```

**Arguments****win**

A pointer to the window.

**ch**

The character to be added. A new-line character (`\n`) clears the line to the end, and moves the cursor to the next line at the same x coordinate. A return (`\r`) moves the cursor to the beginning of the line on the window. A tab (`\t`) moves the cursor to the next tabstop within the window.

**Description**

When the `waddch` function is used on a subwindow, it writes the character onto the underlying window as well.

The `addch` routine performs the same function as `waddch`, but on the `stdscr` window.

The cursor is moved after the character is written to the screen.

**Return Values**

OK

Indicates success.

ERR

Indicates that writing the character would cause the screen to scroll illegally. For more information, see the `scrollok` function in this section.

## [w]addstr

---

## [w]addstr

Add the string pointed to by *str* to the window at the current position of the cursor.

### Format

```
#include <curses.h>

int addstr (char *str);

int waddstr (WINDOW *win, char *str);
```

### Arguments

**win**

A pointer to the window.

**str**

A pointer to a character string.

### Description

When the `waddstr` function is used on a subwindow, the string is written onto the underlying window as well.

The `addstr` routine performs the same function as `waddstr`, but on the `stdscr` window.

The cursor position changes as a result of calling this routine.

### Return Values

OK

Indicates success.

ERR

Indicates that the function causes the screen to scroll illegally, but it places as much of the string onto the window as possible. For more information, see the `scrollok` function in this section.

---

## alarm

Sends the signal SIGALRM (defined in the <signal.h> header file) to the invoking process after the number of seconds indicated by its argument has elapsed.

### Format

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

```
unsigned int alarm (unsigned int seconds); (ISO POSIX-1)
```

```
int alarm (unsigned int seconds); (Compatibility)
```

### Argument

#### **seconds**

Has a maximum limit of LONG\_MAX seconds.

### Description

Calling the alarm function with a 0 argument cancels any pending alarms.

Unless it is caught or ignored, the signal generated by alarm terminates the process. Successive alarm calls reinitialize the alarm clock. Alarms are not stacked.

Because the clock has a 1-second resolution, the signal may occur up to 1 second early. If the SIGALRM signal is caught, resumption of execution may be held up due to scheduling delays.

When the SIGALRM signal is generated, a call to SYS\$WAKE is generated whether or not the process is hibernating. The pending wake causes the current pause() to return immediately (after completing any function that catches the SIGALRM).

### Return Value

|   |                                                                |
|---|----------------------------------------------------------------|
| n | The number of seconds remaining from a previous alarm request. |
|---|----------------------------------------------------------------|

---

## asctime, asctime\_r

Converts a broken-down time in a `tm` structure into a 26-character string in the following form:

```
Sun Sep 16 01:03:52 1984\n\0
```

All fields have a constant width.

### Format

```
#include <time.h>
```

```
char *asctime (const struct tm *timeptr);
```

```
char *asctime_r (const struct tm *timeptr, char *buffer); (ISO POSIX-1)
```

### Arguments

#### **timeptr**

A pointer to a structure of type `tm`, which contains the broken-down time.

The `tm` structure is defined in the `<time.h>` header file, and also shown in Table REF-4 in the description of `localtime`.

#### **buffer**

A pointer to a character array that is at least 26 bytes long. This array is used to store the generated date-and-time string.

### Description

The `asctime` and `asctime_r` functions convert the contents of `tm` into a 26-character string and returns a pointer to the string.

The difference between `asctime_r` and `asctime` is that the former puts the result into a user-specified buffer. The latter puts the result into thread-specific static memory allocated by the *Compaq C* RTL, which can be overwritten by subsequent calls to `ctime` or `asctime`; you must make a copy if you want to save it.

On success, `asctime` returns a pointer to the string; `asctime_r` returns its second argument. On failure, these functions return the `NULL` pointer.

See the `localtime` function in this section for a list of the members in `tm`.

---

#### **Note**

---

Generally speaking, UTC-based time functions can affect in-memory time-zone information, which is process-wide data. However, if the system time zone remains the same during the execution of the application (which is the common case) and the cache of timezone files is enabled (which is the default), then the `_r` variant of the time functions `asctime_r`, `ctime_r`, `gmtime_r` and `localtime_r`, is both thread-safe and AST-reentrant.

If, however, the system time zone can change during the execution of the application or the cache of timezone files is not enabled, then both variants of the UTC-based time functions belong to the third class of functions, which are neither thread-safe nor AST-reentrant.

---



## **Return Values**

|      |                                         |
|------|-----------------------------------------|
| x    | A pointer to the string, if successful. |
| NULL | Indicates failure.                      |

## asin

---

## asin

Returns the arc sine of its argument.

### Format

```
#include <math.h>

double asin (double x);
float asinf (float x); (Alpha only)
long double asinl (long double x); (Alpha only)
double asind (double x); (Alpha only)
float asindf (float x); (Alpha only)
long double asindl (long double x); (Alpha only)
```

### Argument

**x**

A radian expressed as a real number in the domain  $[-1,1]$ .

### Description

The asin functions compute the principal value of the arc sine of  $x$  in the range  $[-\pi/2, \pi/2]$  radians for  $x$  in the domain  $[-1,1]$ .

The asind functions compute the principal value of the arc sine of  $x$  in the range  $[-90,90]$  degrees for  $x$  in the domain  $[-1,1]$ .

When  $\text{abs}(x)$  is greater than 1.0, the value of  $\text{asin}(x)$  is 0, and `errno` is set to `EDOM`.

---

**asinh** *(Alpha only)*

Returns the hyperbolic arc sine of its argument.

**Format**

```
#include <math.h>

double asinh (double x);
float asinhf (float x);
long double asinhl (long double x);
```

**Argument**

**x**

A radian expressed as a real value in the domain  $[-\text{Infinity}, +\text{Infinity}]$ .

**Description**

The asinh functions return the hyperbolic arc sine of  $x$  for  $x$  in the domain  $[-\text{Infinity}, +\text{Infinity}]$ , where  $\text{asinh}(x) = \ln(x + \sqrt{x^2 + 1})$ .

The asinh function is the inverse function of sinh where  $\text{asinh}(\sinh(x)) = x$ .

## assert

---

## assert

Used for implementing run-time diagnostics in programs.

### Format

```
#include <assert.h>

void assert (int expression);
```

### Argument

#### **expression**

An expression that has an int type.

### Description

When `assert` is executed, if *expression* is false (that is, it evaluates to 0), `assert` writes information about the particular call that failed (including the text of the argument, the name of the source file, and the source line number; the latter two are, respectively, the values of the preprocessing macros `__FILE__` and `__LINE__`) to the standard error file in an implementation-defined format. Then, it calls the `abort` function.

The `assert` function writes a message in the following form:

Assertion failed: *expression*, file *aaa*, line *nnn*

If *expression* is true (that is, it evaluates to nonzero) or if the signal `SIGABRT` is being ignored, `assert` returns no value.

---

#### **Note**

---

If a null character (`'\0'`) is part of the expression being asserted, then only the text up to and including the null character is printed, since the null character effectively terminates the string being output.

---

Compiling with the `CC` command qualifier `/DEFINE=NDEBUG` or with the preprocessor directive `#define NDEBUG` ahead of the `#include assert` statement causes the `assert` function to have no effect.

### Example

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

main()
{
 printf("Only this and the assert\n");
 assert(1 == 2); /* expression is FALSE */

 /* abort should be called so the printf will not happen. */
 printf("FAIL abort did not execute");
}
```

---

## atan

### Format

```
#include <math.h>

double atan (double x);
float atanf (float x); (Alpha only)
long double atanl (long double x); (Alpha only)
double atand (double x); (Alpha only)
float atandf (float x); (Alpha only)
long double atandl (long double x); (Alpha only)
```

### Argument

**x**  
A radian expressed as a real number.

### Description

The atan functions compute the principal value of the arc tangent of  $x$  in the range  $[-\pi/2, \pi/2]$  radians.

The atand functions compute the principal value of the arc tangent of  $x$  in the range  $[-90, 90]$  degrees.

---

## atan2

### Format

```
#include <math.h>

double atan2 (double y, double x);
float atan2f (float y, float x); (Alpha only)
long double atan2l (long double y, long double x); (Alpha only)
double atand2 (double y, double x); (Alpha only)
float atand2f (float y, float x); (Alpha only)
long double atand2l (long double y, long double x); (Alpha only)
```

### Arguments

**y**  
A radian expressed as a real number.

**x**  
A radian expressed as a real number.

### Description

The atan2 functions compute the principal value of the arc tangent of  $y/x$  in the range  $[-\pi, \pi]$  radians. The sign of atan2 and atan2f is determined by the sign of  $y$ . The value of atan2( $y, x$ ) is computed as follows, where  $f$  is the number of fraction bits associated with the data type:

| Value of Input Arguments          | Angle Returned                              |
|-----------------------------------|---------------------------------------------|
| $x = 0$ or $y/x > 2^{*(f+1)}$     | $\pi/2 * (\text{sign } y)$                  |
| $x > 0$ and $y/x \leq 2^{*(f+1)}$ | $\text{atan}(y/x)$                          |
| $x < 0$ and $y/x \leq 2^{*(f+1)}$ | $\pi * (\text{sign } y) + \text{atan}(y/x)$ |

The atand2 functions compute the principal value of the arc tangent of  $y/x$  in the range  $[-180, 180]$  degrees. The sign of atand2 and atand2f is determined by the sign of  $y$ .

The following are invalid arguments for the atan2 and atand2 functions:

| Function                 | Exceptional Argument          |
|--------------------------|-------------------------------|
| atan2, atan2f, atan2l    | $x = y = 0$                   |
| atan2, atan2f, atan2l    | $ x  =  y  = \text{Infinity}$ |
| atand2, atand2f, atand2l | $x = y = 0$                   |
| atand2, atand2f, atand2l | $ x  =  y  = \text{Infinity}$ |

---

**atanh** *(Alpha only)*

Returns the hyperbolic arc tangent of its argument.

**Format**

```
#include <math.h>

double atanh (double x);
float atanhf (float x);
long double atanhf (long double x);
```

**Argument**

**x**

A radian expressed as a real value in the domain  $[-1,1]$ .

**Description**

The atanh functions return the hyperbolic arc tangent of  $x$ . The atanh function is the inverse function of tanh where  $\text{atanh}(\tanh(x)) = x$ .

$|x| > 1$  is an invalid argument.

## atexit

---

## atexit

Registers a function that is called without arguments at program termination.

### Format

```
#include <stdlib.h>

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

### Argument

**func**  
A pointer to the function to be registered.

### Return Values

|         |                                                |
|---------|------------------------------------------------|
| 0       | Indicates that the registration has succeeded. |
| nonzero | Indicates failure.                             |

### Restriction

The `longjmp` function cannot be executed from within the handler, because the destination address of the `longjmp` no longer exists.

### Example

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

static void hw(void);

main()
{
 atexit(hw);
}

static void hw()
{
 puts("Hello, world\n");
}
```

Running this example produces the following output:

```
Hello, world
```



---

## atof

Converts an ASCII character string to a double-precision number.

### Format

```
#include <stdlib.h>

double atof (const char *nptr);
```

### Argument

**nptr**

A pointer to the character string to be converted to a double-precision number. The string is interpreted by the same rules that are used to interpret floating constants.

### Description

The string to be converted has the following format:

[white-spaces][+ | -]digits[radix-character][digits][e | E[+ | -]integer]

Where *radix-character* is defined in the current locale.

The first unrecognized character ends the conversion.

This function is equivalent to `strtod(nptr, (char**) NULL)`.

### Return Values

|                |                                                                                                                                                                     |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x              | The converted value.                                                                                                                                                |
| 0              | Indicates an underflow or the conversion could not be performed. The function sets <code>errno</code> to <code>ERANGE</code> or <code>EINVAL</code> , respectively. |
| $\pm$ HUGE_VAL | Overflow occurred; <code>errno</code> is set to <code>ERANGE</code> .                                                                                               |

## atoi, atol

---

## atoi, atol

Convert strings of ASCII characters to the appropriate numeric values.

### Format

```
#include <stdlib.h>

int atoi (const char *nptr);
long int atol (const char *nptr);
```

### Argument

#### **nptr**

A pointer to the character string to be converted to a numeric value.

### Description

The `atoi` and `atol` functions convert the initial portion of a string to its decimal `int` or `long int` value, respectively. The `atoi` and `atol` functions do not account for overflows resulting from the conversion. The string to be converted has the following format:

[white-spaces][+|-]digits

The function call `atol (str)` is equivalent to `strtol (str, (char**)NULL, 10)`, and the function call `atoi (str)` is equivalent to `(int) strtol (str, (char**)NULL, 10)`, except, in both cases, for the behavior on error.

### Return Value

|   |                      |
|---|----------------------|
| n | The converted value. |
|---|----------------------|

---

## **atoq, atoll** *(Alpha only)*

Convert strings of ASCII characters to the appropriate numeric values. `atoll` is a synonym for `atoq`.

### **Format**

```
#include <stdlib.h>

__int64 atoq (const char *nptr);
__int64 atoll (const char *nptr);
```

### **Argument**

**nptr**

A pointer to the character string to be converted to a numeric value.

### **Description**

The `atoq` (or `atoll`) function converts the initial portion of a string to its decimal `__int64` value. This function does not account for overflows resulting from the conversion. The string to be converted has the following format:

`[white-spaces][+|-]digits`

The function call `atoq (str)` is equivalent to `strtoq (str, (char**)NULL, 10)`, except for the behavior on error.

### **Return Value**

|                |                      |
|----------------|----------------------|
| <code>n</code> | The converted value. |
|----------------|----------------------|

## basename

---

## basename

Returns the last component of a pathname.

### Format

```
#include <libgen.h>

char *basename (char *path);
```

### Function Variants

The `basename` function has variants named `_basename32` and `_basename64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Argument

#### **path**

A UNIX style pathname from which the base pathname is extracted.

### Description

The `basename` function takes the UNIX style pathname pointed to by *path* and returns a pointer to the pathname's final component, deleting any trailing slash (/) characters.

If *path* consists entirely of the slash (/) character, the function returns a pointer to the string "/".

If *path* is a NULL pointer or points to an empty string, the function returns a pointer to the string ".".

The `basename` function can modify the string pointed to by *path*.

### Return Values

|     |                                                                |
|-----|----------------------------------------------------------------|
| x   | A pointer to the final component of <i>path</i> .              |
| "/" | If <i>path</i> consists entirely of the '/' character.         |
| ."  | If <i>path</i> is a NULL pointer or points to an empty string. |

---

## bcmp

Compares byte strings.

### Format

```
#include <strings.h>

void bcmp (const void *string1, const void *string2, size_t length);
```

### Arguments

**string1, string2**

The byte strings to be compared.

**length**

The length (in bytes) of the strings.

### Description

The `bcmp` function compares the byte string in *string1* against the byte string in *string2*.

Unlike the string functions, there is no checking for null bytes. Zero-length strings are always identical.

Note that `bcmp` is equivalent to `memcmp`, which is defined by the ANSI C Standard. Therefore, using `memcmp` is recommended for portable programs.

### Return Values

|         |                                |
|---------|--------------------------------|
| 0       | The strings are identical.     |
| Nonzero | The strings are not identical. |

## bcopy

---

## bcopy

Copies byte strings.

### Format

```
#include <strings.h>
```

```
void bcopy (const void *source, void *destination, size_t length);
```

### Arguments

**source**

Pointer to the source string.

**destination**

Pointer to the destination string.

**length**

The length (in bytes) of the string.

### Description

The `bcopy` function operates on variable-length strings of bytes. It copies the value of the *length* argument in bytes from the string in the *source* argument to the string in the *destination* argument.

Unlike the string functions, there is no checking for null bytes. If the *length* argument is 0 (zero), no bytes are copied.

Note that `bcopy` is equivalent to `memcpy`, which is defined by the ANSI C Standard. Therefore, using `memcpy` is recommended for portable programs.

---

## box

Draws a box around the window using the character *vert* as the character for drawing the vertical lines of the rectangle, and *hor* for drawing the horizontal lines of the rectangle.

### Format

```
#include <curses.h>

int box (WINDOW *win, char vert, char hor);
```

### Arguments

**win**

The address of the window.

**vert**

The character for the vertical edges of the window.

**hor**

The character for the horizontal edges of the window.

### Description

The box function copies boxes drawn on subwindows onto the underlying window. Use caution when using functions such as *overlay* and *overwrite* with boxed subwindows. Such functions copy the box onto the underlying window.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

## brk

---

## brk

Determines the lowest virtual address that is not used with the program.

### Format

```
#include <stdlib.h>

void *brk (unsigned long int addr);
```

### Argument

#### **addr**

The lowest address, which the function rounds up to the next multiple of the page size. This rounded address is called the *break address*.

### Description

An address that is greater than or equal to the break address and less than the stack pointer is considered to be outside the program's address space. Attempts to reference it will cause access violations.

When a program is executed, the break address is set to the highest location defined by the program and data storage areas. Consequently, brk is needed only by programs that have growing data areas.

### Return Values

|              |                                                                                                                        |
|--------------|------------------------------------------------------------------------------------------------------------------------|
| n            | The new break address.                                                                                                 |
| (void *)(-1) | Indicates that the program is requesting too much memory. <code>errno</code> and <code>vaxc\$errno</code> are updated. |

### Restriction

Unlike other C library implementations, the *Compaq C* RTL memory allocation functions (such as `malloc`) do not rely on `brk` or `sbrk` to manage the program heap space. Consequently, on OpenVMS systems, calling `brk` or `sbrk` can interfere with memory allocation routines. The `brk` and `sbrk` functions are provided only for compatibility purposes.



---

## bsearch

Performs a binary search. It searches an array of sorted objects for a specified object.

### Format

```
#include <stdlib.h>

void *bsearch (const void *key, const void *base, size_t nmemb, size_t size, int (*compar)
 (const void *, const void *));
```

### Function Variants

The `bsearch` function has variants named `_bsearch32` and `_bsearch64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**key**

A pointer to the object to be sought in the array. This pointer should be of type pointer-to-object and cast to type pointer-to-void.

**base**

A pointer to the initial member of the array. This pointer should be of type pointer-to-object and cast to type pointer-to-void.

**nmemb**

The number of objects in the array.

**size**

The size of an object, in bytes.

**compar**

A pointer to the comparison function.

### Description

The array must first be sorted in increasing order according to the specified comparison function pointed to by *compar*.

Two arguments are passed to the comparison function pointed to by *compar*. The two arguments point to the objects being compared. Depending on whether the first argument is less than, equal to, or greater than the second argument, the comparison function must return an integer less than, equal to, or greater than 0.

It is not necessary for the comparison function (*compar*) to compare every byte in the array. Therefore, the objects in the array can contain arbitrary data in addition to the data being compared.

Since it is declared as type pointer-to-void, the value returned must be cast or assigned into type pointer-to-object.

## bsearch

### Return Values

|      |                                                                                       |
|------|---------------------------------------------------------------------------------------|
| x    | A pointer to the matching member of the array or a null pointer if no match is found. |
| NULL | Indicates that the key cannot be found in the array.                                  |

### Example

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

#define SSIZE 30

extern int compare(); /* prototype for comparison function */

int array[SSIZE] = {30, 1, 29, 2, 28, 3, 27, 4, 26, 5,
 24, 6, 23, 7, 22, 8, 21, 9, 20, 10,
 19, 11, 18, 12, 17, 13, 16, 14, 15, 25};

/* This program takes an unsorted array, sorts it using qsort, */
/* and then calls bsearch for each element in the array, */
/* making sure that bsearch returns the correct element. */
main()
{
 int i;
 int failure = FALSE;
 int *rkey;

 qsort(array, SSIZE, sizeof (array[0]), &compare);

 /* search for each element */
 for (i = 0; i < SSIZE - 1; i++) {
 /* search array element i */
 rkey = bsearch((array + i), array, SSIZE,
 sizeof(array[0]), &compare);
 /* check for successful search */
 if (&array[i] != rkey) {
 printf("Not in array, array element %d\n", i);
 failure = TRUE;
 break;
 }
 }
 if (!failure)
 printf("All elements successfully found!\n");
}

/* Simple comparison routine. */
/* Returns: = 0 if a == b */
/* < 0 if a < b */
/* > 0 if a > b */
int compare(int *a, int *b)
{
 return (*a - *b);
}
```

This example program outputs the following:

All elements successfully found!

---

## btowc

Converts a one-byte multibyte character to a wide character in the initial shift state.

### Format

```
#include <wchar.h>
wint_t btowc (int c);
```

### Argument

**c**  
The character to be converted to a wide-character representation.

### Description

The `btowc` function determines whether `(unsigned char)c` is a valid one-byte multibyte character in the initial shift state, and if so, returns a wide-character representation of that character.

### Return Values

|                   |                                                                                                                                                                  |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>    | The wide-character representation of unsigned char <i>c</i> .                                                                                                    |
| <code>WEOF</code> | Indicates an error. The <i>c</i> argument has the value <code>EOF</code> or does not constitute a valid one-byte multibyte character in the initial shift state. |

## bzero

---

## bzero

Copies null characters into byte strings.

### Format

```
#include <strings.h>

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

### Arguments

**string**

Specifies the byte string into which you want to copy null characters.

**length**

Specifies the length (in bytes) of the string.

### Description

The `bzero` function copies null characters (`'\0'`) into the byte string pointed to by *string* for *length* bytes. If *length* is 0 (zero), then no bytes are copied.

---

## cabs

Returns the absolute value of a complex number.

### Format

```
#include <math.h>

double cabs (cabs_t z);
float cabsf (cabsf_t z); (Alpha only)
long double cabsl (cabsl_t z); (Alpha only)
```

### Argument

**z**

A structure of type `cabs_t`, `cabsf_t`, or `cabsl_t`. These types are defined in the `<math.h>` header file as follows:

```
typedef struct {double x,y;} cabs_t;
typedef struct { float x, y; } cabsf_t;
(Alpha only)
typedef struct { long double x, y; } cabsl_t;
(Alpha only)
```

### Description

The `cabs` functions return the absolute value of a complex number by computing the Euclidean distance between its two points as the square root of their respective squares:

$$\sqrt{x^2 + y^2}$$

On overflow, the return value is undefined.

The `cabs`, `cabsf`, and `cabsl` functions are equivalent to the `hypot`, `hypotf`, and `hypotl` functions, respectively.

## calloc

---

## calloc

Allocates an area of zeroed memory. This function is AST-reentrant.

### Format

```
#include <stdlib.h>

void *calloc (size_t number, size_t size);
```

### Function Variants

The calloc function has variants named `_calloc32` and `_calloc64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**number**  
The number of items to be allocated.

**size**  
The size of each item.

### Description

The calloc function initializes the items to 0.

See also malloc and realloc in this section.

### Return Values

|      |                                                                         |
|------|-------------------------------------------------------------------------|
| n    | The address of the first byte, which is aligned on a quadword boundary. |
| NULL | Indicates an inability to allocate the space.                           |

---

## catclose

Closes a message catalog.

### Format

```
#include <nl_types.h>
int catclose (nl_catd catd);
```

### Argument

**catd**

A message catalog descriptor. This is returned by a successful call to *catopen*.

### Description

The *catclose* function closes the message catalog referenced by *catd* and frees the catalog file descriptor.

### Return Values

- |    |                                                                                                |
|----|------------------------------------------------------------------------------------------------|
| 0  | Indicates that the catalog was successfully closed.                                            |
| -1 | Indicates that an error occurred. The function sets <i>errno</i> to the following value:       |
|    | <ul style="list-style-type: none"><li>• EBADF – The catalog descriptor is not valid.</li></ul> |

---

## catgets

Retrieves a message from a message catalog.

### Format

```
#include <nl_types.h>

char *catgets (nl_catd catd, int set_id, int msg_id, const char *s);
```

### Function Variants

The `catgets` function has variants named `_catgets32` and `_catgets64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**catd**

A message catalog descriptor. This is returned by a successful call to `catopen`.

**set\_id**

An integer set identifier.

**msg\_id**

An integer message identifier.

**s**

A pointer to a default message string that is returned by the function if the message cannot be retrieved.

### Description

The `catgets` function retrieves a message identified by `set_id` and `msg_id`, in the message catalog `catd`. The message is stored in a message buffer in the `nl_catd` structure, which is overwritten by subsequent calls to `catgets`. If a message string needs to be preserved, it should be copied to another location by the program.

### Return Values

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x | Pointer to the retrieved message.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| s | Pointer to the default message string. Indicates that the function is not able to retrieve the requested message from the catalogue. This condition can arise if the requested pair ( <code>set_id</code> , <code>msg_id</code> ) does not represent an existing message from the open catalogue, or it indicates that an error occurred. If an error occurred, the function sets <code>errno</code> to one of the following values: <ul style="list-style-type: none"><li>• EBADF – The catalog descriptor is not valid.</li><li>• EVMSRR – A VMS I/O read error; the VMS error code can be found in <code>vaxc\$errno</code>.</li></ul> |



## Example

```
#include <nl_types.h>
#include <locale.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

/* This test makes use of all the message catalog routines. catopen() */
/* opens the catalog ready for reading, then each of the three */
/* messages in the catalog are extracted in turn using catgets() and */
/* printed out. catclose() closes the catalog after use. */
/* The catalog source file used to create the catalog is as follows: */
/* $ this is a message file
* $
* $quote "
* $ another comment line
* $set 1
* 1 "First set, first message"
* 2 "second message - This long message uses a backslash \
* for continuation."
* $set 2
* 1 "Second set, first message" */

char *default_msg = "this is the first message.";

main()
{
 nl_catd catalog;
 int msg1,
 msg2,
 retval;

 char *cat = "sys$disk:[]catgets_example.cat"; /*Force local catalog*/
 char *msgtxt;
 char string[128];

 /* Create the message test catalog */
 system("gencat catgets_example.msgx catgets_example.cat") ;
 if ((catalog = catopen(cat, 0)) == (nl_catd) - 1) {
 perror("catopen");
 exit(EXIT_FAILURE);
 }

 msgtxt = catgets(catalog, 1, 1, default_msg);
 printf("%s\n", msgtxt);

 msgtxt = catgets(catalog, 1, 2, default_msg);
 printf("%s\n", msgtxt);

 msgtxt = catgets(catalog, 2, 1, default_msg);
 printf("%s\n", msgtxt);

 if ((retval = catclose(catalog)) == -1) {
 perror("catclose");
 exit(EXIT_FAILURE);
 }

 delete("catgets_example.cat;") ; /* Remove the test catalog */
}
```

## catgets

Running the example program produces the following result:

```
First set, first message
second message - This long message uses a backslash for continuation.
Second set, first message
```

---

# catopen

Opens a message catalog.

## Format

```
#include <nl_types.h>

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

## Arguments

- name**  
The name of the message catalog to open.
- oflag**  
An object of type `int` that determines whether the locale set for the `LC_MESSAGES` category in the current program's locale or the logical name `LANG` is used to search for the catalog file.

## Description

The `catopen` function opens the message catalog identified by *name*.

If *name* contains a colon (:), a square opening bracket ([), or an angle bracket (<), or is defined as a logical name, then it is assumed that *name* is the complete file specification of the catalog.

If it does not include these characters, `catopen` assumes that *name* is a logical name pointing to an existing catalog file. If *name* is not a logical name, then the logical name `NLSPATH` is used to define the file specification of the message catalog. `NLSPATH` is defined in the user's process. If the `NLSPATH` logical name is not defined, or no message catalog can be opened in any of the components specified by the `NLSPATH`, then the `SYS$NLSPATH` logical name is used to search for a message catalog file.

Both `NLSPATH` and `SYS$NLSPATH` is a comma-separated list of templates. The `catopen` function uses each template to construct a file specification. For example, `NLSPATH` could be defined as:

```
DEFINE NLSPATH SYS$SYSROOT:[SYS$I18N.MSG]%N.CAT,SYS$COMMON:[SYSMSG]%N.CAT
```

In this example, `catopen` first searches the directory `SYS$SYSROOT:[SYS$I18N.MSG]` for the named catalog. If the named catalog is not found there, the directory `SYS$COMMON:[SYSMSG]` is searched. The catalog name is constructed by substituting `%N` with the name passed to `catopen`, and adding the `.cat` suffix. `%N` is known as a substitution field. The following substitution fields are valid:

| Field | Meaning                                                   |
|-------|-----------------------------------------------------------|
| %N    | Substitute the <i>name</i> passed to <code>catopen</code> |

| Field           | Meaning                                                                                                                                                                                                                                                |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| %L <sup>1</sup> | Substitute the locale name.<br>The period (.) and at-sign (@) characters in the locale name are replaced by an underscore (_) character.<br>For example, the "zh_CN.dechanzi@radical" locale name results in a substitution of ZH_CN_DECHANZI_RADICAL. |
| %l <sup>1</sup> | Substitute the <i>language</i> part of the locale name. For example, the language part of the en_GB.ISO8859-1 locale name is en.                                                                                                                       |
| %t <sup>1</sup> | Substitute the <i>territory</i> part of the locale name. For example, the territory part of the en_GB.ISO8859-1 locale is GB.                                                                                                                          |
| %c <sup>1</sup> | Substitute the <i>codeset</i> name from the locale name. For example, the codeset name of the en_GB.ISO8859-1 locale name is ISO8859-1.                                                                                                                |

---

<sup>1</sup>This substitution assumes that the locale name is of the form *language\_territory.codeset@mode*

---

If the *oflag* argument is set to NL\_CAT\_LOCALE, then the current locale as defined for the LC\_MESSAGES category is used to determine the substitution for the %L, %l, %t, and %c substitution fields. If the *oflag* argument is set to 0, then the value of the LANG environment variable is used as a locale name to determine the substitution for these fields. Note that using NL\_CAT\_LOCALE conforms to the XPG4 specification while a value of 0 (zero) exists for the purpose of preserving XPG3 compatibility. Note also, that catopen uses the value of the LANG environment variable without checking whether the program's locale can be set using this value. That is, catopen does not check whether this value can serve as a valid locale argument in the setlocale call.

If the substitution value is not defined, an empty string is substituted.

A leading comma or two adjacent commas (,,) is equivalent to specifying %N. For example,

```
DEFINE NLSPATH ",%N.CAT,SYS$COMMON:[SYSMSG.%L]%N.CAT"
```

In this example, catopen searches in the following locations in the order shown:

1. *name* (in the current directory)
2. *name.cat* (in the current directory)
3. SYS\$COMMON:[SYSMSG.*locale\_name*]*name.cat*

## Return Values

|   |                                                                                                                                 |
|---|---------------------------------------------------------------------------------------------------------------------------------|
| x | A message catalog file descriptor. Indicates the call was successful. This descriptor is used in calls to catgets and catclose. |
|---|---------------------------------------------------------------------------------------------------------------------------------|

(nl\_catd) –1

Indicates an error occurred. The function sets *errno* to one of the following values:

- EACCES – Insufficient privilege or file protection violation, or file currently locked by another user.
- EMFILE – Process channel count exceeded.
- ENAMETOOLONG – The full file specification for message catalogue is too long
- ENOENT – Unable to find the requested message catalogue.
- ENOMEM – Insufficient memory available.
- ENOTDIR – Part of the *name* argument is not a valid directory.
- EVMSEERR – An error occurred that does not match any *errno* value. Check the value of *vaxc\$errno*.

**cbrt** *(Alpha only)*

---

**cbrt** *(Alpha only)*

Returns the rounded cube root of  $y$ .

### Format

```
#include <math.h>

double cbrt (double y);
float cbrtf (float y);
long double cbrtl (long double y);
```

### Argument

**y**  
A real number.

---

## ceil

Returns the smallest integer that is greater than or equal to its argument.

### Format

```
#include <math.h>

double ceil (double x);
float ceilf (float x); (Alpha only)
long double ceill (long double x); (Alpha only)
```

### Argument

**x**  
A real value.

### Return Value

**n**                      The smallest integer greater than or equal to the function argument.

## **cfree**

---

## **cfree**

Makes available for reallocation the area allocated by a previous `calloc`, `malloc`, or `realloc` call. This function is AST-reentrant.

### **Format**

```
#include <stdlib.h>

void cfree (void *ptr);
```

### **Argument**

**ptr**

The address returned by a previous call to `malloc`, `calloc`, or `realloc`.

### **Description**

The contents of the deallocated area are unchanged.

In *Compaq C* for *OpenVMS* Systems, the `free` and `cfree` functions are equivalent. Some other C implementations use `free` with `malloc` or `realloc`, and `cfree` with `calloc`. However, since the ANSI C standard does not include `cfree`, using `free` may be preferable.

See also `free` in this section.



---

## chdir

Changes the default directory.

### Format

```
#include <unistd.h>

int chdir (const char *dir_spec); (ISO POSIX-1)

int chdir (const char *dir_spec, ...); (COMPAQ C Extension)
```

### Arguments

#### dir\_spec

A null-terminated character string naming a directory in either an OpenVMS or UNIX style specification.

...

This argument is a *Compaq C* extension available when not defining any of the standards-related feature-test macros (see Section 1.5 and not compiling in strict ANSI C mode (/STANDARD=ANSI89). The argument is an optional flag of type int that is significant only when calling chdir from USER mode.

If the value of the flag is 1, the new directory is effective across images. If the value is not 1, the original default directory is restored when the image exits.

### Description

The chdir function changes the default directory. The change can be permanent or temporary. Permanent means that the new directory remains as the default directory after the image exits. Temporary means that on image exit, the default is set to whatever it was before the execution of the image.

There are two ways of making the change permanent:

- Call chdir from USER mode with the second argument set to 1.
- Call chdir from SUPERVISOR or EXECUTIVE mode, regardless of the value of the second argument.

Otherwise, the change is temporary.

### Return Values

|    |                                                                         |
|----|-------------------------------------------------------------------------|
| 0  | Indicates that the directory is successfully changed to the given name. |
| -1 | Indicates that the change attempt has failed.                           |

## chmod

---

## chmod

Changes the file protection of a file.

### Format

```
#include <stat.h>

int chmod (const char *file_spec, mode_t mode);
```

### Arguments

#### **file\_spec**

The name of an OpenVMS or UNIX style file specification.

#### **mode**

A file protection. Modes are constructed by performing a bitwise OR on any of the values shown in Table REF–2.

**Table REF–2 File Protection Values and Their Meanings**

| Value | Privilege     |
|-------|---------------|
| 0400  | OWNER:READ    |
| 0200  | OWNER:WRITE   |
| 0100  | OWNER:EXECUTE |
| 0040  | GROUP:READ    |
| 0020  | GROUP:WRITE   |
| 0010  | GROUP:EXECUTE |
| 0004  | WORLD:READ    |
| 0002  | WORLD:WRITE   |
| 0001  | WORLD:EXECUTE |

When you supply a *mode* value of 0, the chmod function gives the file the user's default file protection.

The system is given the same privileges as the owner. A WRITE privilege also implies a DELETE privilege.

### Description

You must have a WRITE privilege for the file specified to change the mode.

### Return Values

|    |                                                  |
|----|--------------------------------------------------|
| 0  | Indicates that the mode is successfully changed. |
| –1 | Indicates that the change attempt has failed.    |

---

## chown

Changes the owner user identification code (UIC) of the file.

### Format

```
#include <unistd.h>

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

### Arguments

**file\_spec**

The address of an ASCII file name.

**owner**

An integer corresponding to the new owner UIC of the file.

**group**

An integer corresponding to the group UIC of the file.

### Return Values

|    |                    |
|----|--------------------|
| 0  | Indicates success. |
| −1 | Indicates failure. |

## [w]clear

---

## [w]clear

Erase the contents of the specified window and reset the cursor to coordinates (0,0). The `clear` function acts on the `stdscr` window.

### Format

```
#include <curses.h>

int clear();

int wclear (WINDOW *win);
```

### Argument

**win**  
A pointer to the window.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

---

**clearerr**

Resets the error and end-of-file indicators for a file (so that `ferror` and `feof` will not return a nonzero value).

**Format**

```
#include <stdio.h>

void clearerr (FILE *file_ptr);
```

**Argument**

**file\_ptr**  
A file pointer.

## clearok

---

## clearok

Sets the clear flag for the window.

### Format

```
#include <curses.h>

clearok (WINDOW *win, bool boolf);
```

### Arguments

#### **win**

The entire size of the terminal screen. You can use the windows `stdscr` and `curscr` with `clearok`.

#### **boolf**

A Boolean value of `TRUE` or `FALSE`. If the argument is `TRUE`, this forces a clearscreen to be printed on the next call to `refresh`, or stops the screen from being cleared if *boolf* is `FALSE`.

The type `bool` is defined in the `<curses.h>` header file as follows:

```
#define bool int
```

### Description

Unlike the `clear` function, the `clearok` function does not alter the contents of the window. If the *win* argument is `curscr`, the next call to `refresh` causes a clearscreen, even if the window passed to `refresh` is not a window the size of the entire terminal screen.

---

## clock

Determines the CPU time (in 10-millisecond units) used since the beginning of the process. The time reported is the sum of the user and system times of the calling process and any terminated child processes for which the calling process has executed wait or system.

### Format

```
#include <time.h>

clock_t clock (void);
```

### Description

The value returned by the clock function must be divided by the value of the CLK\_TCK, as defined in the standard header file <time.h>, to obtain the time in seconds.

The type clock\_t is defined in the <time.h> header file as follows:

```
typedef long int clock_t;
```

Only the accumulated times for child processes running a C main program or a program that calls VAXC\$CRTL\_INIT or DECC\$CRTL\_INIT are included.

A typical usage of the clock function is to call it after a program does its initial setup, and then again after the program executes the code to be timed. Then subtract the two values to give elapsed CPU time.

### Return Values

|    |                                                          |
|----|----------------------------------------------------------|
| n  | The processor time used.                                 |
| -1 | Indicates that the processor time used is not available. |

## close

---

## close

Closes the file associated with a file descriptor.

### Format

```
#include <unistd.h>

int close (int file_desc);
```

### Argument

**file\_desc**  
A file descriptor.

### Description

The close function tries to write buffered data by using an implicit call to fflush. If the write fails (because the disk is full or the user's quota was exceeded, for example), close continues executing. It closes the VMS channel, deallocates any buffers, and releases the memory associated with the file descriptor (or FILE pointer). Any buffered data is lost, and the file descriptor (or FILE pointer) no longer refers to the file.

If your program needs to recover from errors when flushing buffered data, it should make an explicit call to fsync (or fflush) before calling close.

### Return Values

|    |                                                                                                                                                                 |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates that the file is properly closed.                                                                                                                     |
| -1 | Indicates that the file descriptor is undefined or an error occurred while the file was being closed (for example, if the buffered data cannot be written out). |

### Example

```
#include <unistd.h>

int fd;
.
.
.
fd = open ("student.dat", 1);
.
.
.
close(fd);
```



---

## closedir

Closes directories.

### Format

```
#include <dirent.h>

int closedir (DIR *dir_pointer);
```

### Argument

#### **dir\_pointer**

Pointer to the `dir` structure of an open directory.

### Description

The `closedir` function closes a directory stream and frees the structure associated with the *dir\_pointer* argument. Upon return, the value of *dir\_pointer* does not necessarily point to an accessible object of the type `DIR`.

The type `DIR`, which is defined in the `<dirent.h>` header file, represents a directory stream that is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files. You can remove files from or add files to a directory asynchronously to the operation of the `readdir` function.

---

#### Note

---

An open directory must always be closed with the `closedir` function to ensure that the next attempt to open the directory is successful.

---

### Example

The following example shows how to search a directory for the entry name, using the `opendir`, `readdir`, and `closedir` functions:

```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define FOUND 1
#define NOT_FOUND 0

static int dir_example(const char *name, unsigned int unix_style)
{
 DIR *dir_pointer;
 struct dirent *dp;

 if (unix_style)
 dir_pointer = opendir(".");
 else
 dir_pointer = opendir(getenv("PATH"));

 if (!dir_pointer) {
 perror("opendir");
 return NOT_FOUND;
 }
}
```

## closedir

```
/* Note, that if opendir() was called with UNIX style file */
/* spec like ".", readdir() will return only a single */
/* version of each file in the directory. In this case the */
/* name returned in d_name member of the dirent structure */
/* will contain only file name and file extension fields, */
/* both lowercased like "foo.bar". */

/* If opendir() was called with VMS style file spec, */
/* readdir() will return every version of each file in the */
/* directory. In this case the name returned in d_name */
/* member of the dirent structure will contain file name, */
/* file extension and file version fields. All in upper */
/* case, like "FOO.BAR;1". */

for (dp = readdir(dir_pointer);
 dp && strcmp(dp->d_name, name);
 dp = readdir(dir_pointer))
 ;

closedir(dir_pointer);

if (dp != NULL)
 return FOUND;
else
 return NOT_FOUND;
}

int main(void)
{
 char *filename = "foo.bar";
 FILE *fp;

 remove(filename);

 if (!(fp = fopen(filename, "w"))) {
 perror("fopen");
 return (EXIT_FAILURE);
 }

 if (dir_example("FOO.BAR;1", 0) == FOUND)
 puts("VMS style: found");
 else
 puts("VMS style: not found");

 if (dir_example("foo.bar", 1) == FOUND)
 puts("Unix style: found");
 else
 puts("Unix style: not found");

 fclose(fp);
 remove(filename);
 return(EXIT_SUCCESS);
}
```

## Return Values

|    |                                                                  |
|----|------------------------------------------------------------------|
| 0  | Indicates success.                                               |
| -1 | Indicates an error and is further specified in the global errno. |

---

## **[w]clrattr**

Deactivate the video display attribute *attr* within the window. The `clrattr` function acts on the `stdscr` window.

### **Format**

```
#include <curses.h>

int clrattr (int attr);

int wclrattr (WINDOW *win, int attr);
```

### **Arguments**

#### **win**

A pointer to the window.

#### **attr**

Video display attributes that can be blinking, boldface, reverse video, and underlining; they are represented by the defined constants `_BLINK`, `_BOLD`, `_REVERSE`, and `_UNDERLINE`. To clear multiple attributes, separate them with a bitwise OR operator ( `|` ) as follows:

```
clrattr(_BLINK | _UNDERLINE);
```

### **Description**

These functions are specific to *Compaq C* for *OpenVMS* Systems and are not portable.

### **Return Values**

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

## [w]clrtoobot

---

## [w]clrtoobot

Erase the contents of the window from the current position of the cursor to the bottom of the window. The `clrtoobot` function acts on the `stdscr` window.

### Format

```
#include <curses.h>

int clrtoobot();

int wclrtoobot (WINDOW *win);
```

### Argument

**win**  
A pointer to the window.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

---

**[w]clrtoeol**

Erase the contents of the window from the current cursor position to the end of the line on the specified window. The `clrtoeol` function acts on the `stdscr` window.

**Format**

```
#include <curses.h>

int clrtoeol();

int wclrtoeol (WINDOW *win);
```

**Argument**

**win**  
A pointer to the window.

**Return Values**

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

---

**confstr**

Determines the current value of a specified system variable defined by a string value.

**Format**

```
#include <unistd.h>

size_t confstr (int name, char *buf, size_t len);
```

**Arguments****name**

The system variable setting. Valid values for the *name* argument are the `_CS_X` names defined in the `<unistd.h>` header file.

**buf**

Pointer to the buffer where the `confstr` function copies the *name* value.

**len**

The size of the buffer storing the *name* value.

**Description**

The `confstr` function allows an application to determine the current setting of certain system parameters, limits, or options that are defined by a string value. The function is mainly used by applications to find the system default value for the `PATH` environment variable.

If the following conditions are true, then the `confstr` function copies that value into a *len*-byte buffer pointed to by *buf*:

- The *len* argument is not 0 (zero).
- The *name* argument has a system-defined value.
- The *buf* argument is not a NULL pointer.

If the returned string is longer than *len* bytes, including the terminating null, then the `confstr` function truncates the string to *len* - 1 bytes and adds a terminating null to the result. The application can detect that the string was truncated by comparing the value returned by the `confstr` function with the value of the *len* argument.

The `<limits.h>` header file contains system-defined limits. The `<unistd.h>` header file contains system-defined environmental variables.

**Example**

To find out how big a buffer is needed to store the string value of *name*, enter:

```
confstr(_CS_PATH, NULL, (size_t) 0)
```

The `confstr` function returns the size of the buffer necessary.

## Return Values

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | <p>Indicates an error. When the specified <i>name</i> value:</p> <ul style="list-style-type: none"><li>• Is invalid, <code>errno</code> is set to <code>EINVAL</code>.</li><li>• Does not have a system-defined value, <code>errno</code> is not set.</li></ul>                                                                                                                                                                                                                                                                                                                                                                     |
| n | <p>The size of the buffer needed to hold the value.</p> <ul style="list-style-type: none"><li>• When the value of the <i>name</i> argument is system-defined, <code>confstr</code> returns the size of the buffer needed to hold the entire value. If this return value is greater than the <i>len</i> value, the string returned as the <i>buf</i> value is truncated.</li><li>• When the value of the <i>len</i> argument is set to 0 or the <i>buf</i> value is <code>NULL</code>, <code>confstr</code> returns the size of the buffer needed to hold the entire system-defined value. The string value is not copied.</li></ul> |

## **copysign** *(Alpha only)*

---

## **copysign** *(Alpha only)*

Returns  $x$  with the same sign as  $y$ .

### **Format**

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

```
double copysign (double x, double y);
```

```
float copysignf (float x, float y); (Alpha only)
```

```
long double copysignl (long double x, long double y); (Alpha only)
```

### **Arguments**

**x**

A real value.

**y**

A real value.

### **Description**

The `copysign` functions return  $x$  with the same sign as  $y$ . IEEE 754 requires `copysign( $x$ ,NaN)`, `copysignf( $x$ ,NaN)` and `copysignl( $x$ ,NaN)` to return  $+x$  or  $-x$ .

### **Return Values**

**x**

The value of  $x$  with the same sign as  $y$ .



---

**COS**

Returns the cosine of its radian argument.

**Format**

```
#include <math.h>

double cos (double x);
float cosf (float x); (Alpha only)
long double cosl (long double x); (Alpha only)
double cosd (double x); (Alpha only)
float cosdf (float x); (Alpha only)
long double cosdl (long double x); (Alpha only)
```

**Argument**

**x**  
A radian expressed as a real value.

**Description**

The cos functions return the cosine of their argument, measured in radians.

The cosd functions return the cosine of their argument, measured in degrees.

$|x| = \text{Infinity}$  is an invalid argument.

**Return Values**

|          |                                                                   |
|----------|-------------------------------------------------------------------|
| <b>x</b> | The cosine of the argument.                                       |
| HUGE_VAL | Indicates that the argument is too large; errno is set to ERANGE. |

## cosh

---

## cosh

Returns the hyperbolic cosine of its radian argument.

### Format

```
#include <math.h>

double cosh (double x);
float coshf (float x); (Alpha only)
long double coshl (long double x); (Alpha only)
```

### Argument

**x**  
A radian expressed as a real number.

### Description

The cosh functions return the hyperbolic cosine of  $x$  and are defined as  $(e^{**x} + e^{*(-x)})/2$ .

### Return Values

|                 |                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------|
| <b>x</b>        | The hyperbolic cosine of the argument.                                                       |
| <b>HUGE_VAL</b> | Indicates that the argument is too large; <code>errno</code> is set to <code>ERANGE</code> . |

---

## cot

Returns the cotangent of its radian argument.

### Format

```
#include <math.h>

double cot (double x);
float cotf (float x); (Alpha only)
long double cotl (long double x); (Alpha only)
double cotd (double x); (Alpha only)
float cotdf (float x); (Alpha only)
long double cotdl (long double x); (Alpha only)
```

### Argument

**x**  
A radian expressed as a real number.

### Description

The cot functions return the cotangent of their argument, measured in radians.  
The cotd functions return the cotangent of their argument, measured in degrees.  
 $x = 0$  is an invalid argument.

### Return Values

|          |                                                              |
|----------|--------------------------------------------------------------|
| <b>x</b> | The cotangent of the argument.                               |
| HUGE_VAL | Indicates that the argument is zero; errno is set to ERANGE. |

## creat

---

## creat

Creates a new file.

### Format

```
#include <fcntl.h>
```

```
int creat (const char *file_spec, mode_t mode); (ISO POSIX-1)
```

```
int creat (const char *file_spec, mode_t mode, ...); (COMPAQ C Extension)
```

### Arguments

#### **file\_spec**

A null-terminated string containing any valid file specification.

#### **mode**

An unsigned value that specifies the file-protection mode. The compiler performs a bitwise AND operation on the mode and the complement of the current protection mode.

You can construct modes by using the bitwise OR operator ( | ) to create mode combinations. The modes are as follows:

|      |               |
|------|---------------|
| 0400 | OWNER:READ    |
| 0200 | OWNER:WRITE   |
| 0100 | OWNER:EXECUTE |
| 0040 | GROUP:READ    |
| 0020 | GROUP:WRITE   |
| 0010 | GROUP:EXECUTE |
| 0004 | WORLD:READ    |
| 0002 | WORLD:WRITE   |
| 0001 | WORLD:EXECUTE |

The system is given the same privileges as the owner. A WRITE privilege implies a DELETE privilege.

---

#### **Note**

To create files with OpenVMS RMS default protections using the UNIX system-call functions `umask`, `mkdir`, `creat`, and `open`, call `mkdir`, `creat`, and `open` with a file-protection mode argument of 0777 in a program that never specifically calls `umask`. These default protections include correctly establishing protections based on ACLs, previous versions of files, and so on.

In programs that do `vfork/exec` calls, the new process image inherits whether `umask` has ever been called or not from the calling process image. The `umask` setting and whether the `umask` function has ever been called are both inherited attributes.

---

...

An optional argument list of character strings of the following form:

"keyword = value", ... , "keyword = value"

Or in the case of "acc" or "err", this form:

"keyword"

Here, *keyword* is an RMS field in the file access block (FAB) or record access block (RAB); *value* is valid for assignment to that field. Some fields permit you to specify more than one value. In these cases, the values are separated by commas.

The RMS callback keywords "acc" and "err" are the only keywords that do not take values. Instead, they are followed by a pointer to the callback routine to be used, followed by a pointer to a user-specified value to be used as the first argument of the callback routine. For example, to set up an access callback routine called `acc_callback` whose first argument is a pointer to the integer variable `first_arg` in a call to `open`, you can use the following statement:

```
open("file.dat", O_RDONLY, 0, "acc", acc_callback, &first_arg)
```

The second and third arguments to the callback routine must be pointers to a FAB and RAB, respectively, and the routine must have a return type of `int`. If the callback returns a value less than 0, the `open`, `creat`, or `fopen` fails. The error callback can correct the error condition and return a status greater than or equal to 0 to continue the `creat` call. Assuming the previous `open` statement, the function prototype for `acc_callback` would be similar to the following statement:

```
#include <rms.h>

int acc_callback(int *first_arg, struct FAB *fab, struct RAB *rab);
```

FAB and RAB are defined in the `<rms.h>` header file, and the actual pointers passed to the routine are pointers to the RAB and FAB being used to open the file `file.dat`.

If an access callback routine is established, then it will be called in the `open`-type routine immediately before the call to the RMS function `sys$create` or `sys$open`. If an error callback routine is established and an error status is returned from the `sys$create` or `sys$open` function, then the callback routine will be invoked immediately after the status is checked and the error value is discovered.

**Note**

Any manipulation of the RAB or FAB in a callback function could lead to serious problems in later calls to the *Compaq C* RTL I/O functions.

Table REF–3 describes the RMS keywords and values.

**Table REF–3   RMS Valid Keywords and Values**

| Keyword   | Value    | Description              |
|-----------|----------|--------------------------|
| "acc"     | callback | Access callback routine. |
| "alq = n" | decimal  | Allocation quantity.     |

(continued on next page)

Table REF-3 (Cont.) RMS Valid Keywords and Values

| Keyword                   | Value    | Description                                                                                         |
|---------------------------|----------|-----------------------------------------------------------------------------------------------------|
| "bls = n"                 | decimal  | Block size.                                                                                         |
| "ctx = bin"               | string   | No translation of '\n' to the terminal. Use this for writing binary data to files.                  |
| "ctx=cvt"                 | string   | Negates a previous setting of "ctx=nocvt". This is the default.                                     |
| "ctx = nocvt"             | string   | No conversion of Fortran carriage-control bytes.                                                    |
| "ctx = rec"               | string   | Force record mode access.                                                                           |
| "ctx = stm"               | string   | Force stream mode access.                                                                           |
| "ctx=xplct"               | string   | Causes records to be written only when explicitly specified by a call to fflush, fclose, or fclose. |
| "deq = n"                 | decimal  | Default extension quantity.                                                                         |
| "dna = filespec"          | string   | Default file-name string.                                                                           |
| "err"                     | callback | Error callback routine.                                                                             |
| "fop = val, val , . . . " |          | File-processing options:                                                                            |
|                           | ctg      | Contiguous.                                                                                         |
|                           | cbt      | Contiguous-best-try.                                                                                |
|                           | dfw      | Deferred write; only applicable to files opened for shared access.                                  |
|                           | dlt      | Delete file on close.                                                                               |
|                           | tef      | Truncate at end-of-file.                                                                            |
|                           | cif      | Create if nonexistent.                                                                              |
|                           | sup      | Supersede.                                                                                          |
|                           | scf      | Submit as command file on close.                                                                    |
|                           | spl      | Spool to system printer on close.                                                                   |
|                           | tmd      | Temporary delete.                                                                                   |
|                           | tmp      | Temporary (no file directory).                                                                      |
|                           | nef      | Not end-of-file.                                                                                    |
|                           | rck      | Read check compare operation.                                                                       |
|                           | wck      | Write check compare operation.                                                                      |
|                           | mxv      | Maximize version number.                                                                            |
|                           | rwo      | Rewind file on open.                                                                                |
|                           | pos      | Current position.                                                                                   |
|                           | rew      | Rewind file on close.                                                                               |
|                           | sqr      | File can only be processed in a sequential manner.                                                  |
| "fsz = n"                 | decimal  | Fixed header size.                                                                                  |
| "gbc = n"                 | decimal  | The requested number of global buffers for a file.                                                  |
| "mbc = n"                 | decimal  | Multiblock count.                                                                                   |
| "mbf = n"                 | decimal  | Multibuffer count.                                                                                  |
| "mrs = n"                 | decimal  | Maximum record size.                                                                                |

(continued on next page)

Table REF–3 (Cont.) RMS Valid Keywords and Values

| Keyword                 | Value  | Description                                                                                                                                             |
|-------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| "pmt=usr-prmpt"         | string | Prompts for terminal input. Any RMS input from a terminal device will be preceded by "usr-prmpt" when this option and "rop=pmt" are specified.          |
| "rat = val, val . . . " |        | Record attributes:                                                                                                                                      |
|                         | cr     | Carriage-return control.                                                                                                                                |
|                         | blk    | Disallow records to span block boundaries.                                                                                                              |
|                         | ftn    | Fortran print control.                                                                                                                                  |
|                         | none   | Explicitly forces no carriage control.                                                                                                                  |
|                         | prn    | Print file format.                                                                                                                                      |
| "rfm = val"             |        | Record format:                                                                                                                                          |
|                         | fix    | Fixed-length record format.                                                                                                                             |
|                         | stm    | RMS stream record format.                                                                                                                               |
|                         | stmlf  | Stream format with line-feed terminator.                                                                                                                |
|                         | stmcr  | Stream format with carriage-return terminator.                                                                                                          |
|                         | var    | Variable-length record format.                                                                                                                          |
|                         | vfc    | Variable-length record with fixed control.                                                                                                              |
|                         | udf    | Undefined.                                                                                                                                              |
| "rop = val, val . . . " |        | Record-processing operations:                                                                                                                           |
|                         | asy    | Asynchronous I/O.                                                                                                                                       |
|                         | cco    | Cancel Ctrl/O (used with Terminal I/O).                                                                                                                 |
|                         | cvt    | Capitalizes characters on a read from the terminal.                                                                                                     |
|                         | eof    | Positions the record stream to the end-of-file for the connect operation only.                                                                          |
|                         | nlk    | Do not lock record.                                                                                                                                     |
|                         | pmt    | Enables use of the prompt specified by "pmt=usr-prmpt" on input from the terminal.                                                                      |
|                         | pta    | Eliminates any information in the type-ahead buffer on a read from the terminal.                                                                        |
|                         | rea    | Locks record for a read operation for this process, while allowing other accessors to read the record.                                                  |
|                         | rlk    | Locks record for write.                                                                                                                                 |
|                         | rne    | Suppresses echoing of input data on the screen as it is entered on the keyboard.                                                                        |
|                         | rnf    | Indicates that Ctrl/U, Ctrl/R, and DELETE are not to be considered control commands on terminal input, but are to be passed to the application program. |
|                         | rrl    | Reads regardless of lock.                                                                                                                               |

(continued on next page)

Table REF–3 (Cont.) RMS Valid Keywords and Values

| Keyword                  | Value   | Description                                                                                                                         |
|--------------------------|---------|-------------------------------------------------------------------------------------------------------------------------------------|
|                          | syncsts | Returns success status of RMS\$_SYNCH if the requested service completes its task immediately.                                      |
|                          | tmo     | Timeout I/O.                                                                                                                        |
|                          | tpt     | Allows put/write services using sequential record access mode to occur at any point in the file, truncating the file at that point. |
|                          | ulk     | Prohibits RMS from automatically unlocking records.                                                                                 |
|                          | wat     | Wait until record is available, if currently locked by another stream.                                                              |
|                          | rah     | Read ahead.                                                                                                                         |
|                          | wbh     | Write behind.                                                                                                                       |
| “rtv=n”                  | decimal | The number of retrieval pointers that RMS has to maintain in memory (0 to 127,255).                                                 |
| “shr = val, val, . . . ” |         | File sharing options:                                                                                                               |
|                          | del     | Allows users to delete.                                                                                                             |
|                          | get     | Allows users to read.                                                                                                               |
|                          | mse     | Allows multistream connects.                                                                                                        |
|                          | nil     | Prohibits file sharing.                                                                                                             |
|                          | put     | Allows users to write.                                                                                                              |
|                          | upd     | Allows users to update.                                                                                                             |
|                          | upi     | Allows one or more writers.                                                                                                         |
|                          | nql     | No query locking (file level).                                                                                                      |
| “tmo = n”                | decimal | I/O timeout value.                                                                                                                  |

In addition to these options, any option that takes a key value (such as “fop” or “rat”) can be negated by prefixing the value with “no”. For example, specify “fop=notmp” to clear the “tmp” bit in the “fop” field.

---

#### Notes

---

- While these options provide much flexibility and functionality, many of them can also cause severe problems if not used correctly.
  - You cannot share the default *Compaq C* for *OpenVMS* stream file I/O. If you wish to share files, you must specify “ctx=rec” to force record access mode. You must also specify the appropriate “shr” options depending on the type of access you want.
  - If you intend to share a file opened for append, you must specify appropriate share and record-locking options, to allow other accessors to read the record. The reason for doing this: the file is positioned at end-of-file through reading records in a loop until end-of-file is reached.
-



For more information on these options, see the *OpenVMS Record Management Services Reference Manual* manual.

## Description

The *Compaq C* RTL opens the new file for reading and writing, and returns the corresponding file descriptor.

If the file exists:

- A version number one greater than any existing version is assigned to the newly created file.
- By default, the new file inherits certain attributes from the existing version of the file unless those attributes are specified in the `creat` call. The following attributes are inherited:
  - Record format (`fab$b_rfm`)
  - Maximum record size (`fab$w_mrs`)
  - Carriage control (`fab$b_rat`)
  - File protection

If the file did not previously exist:

- It is given the file protection that results from performing a bitwise AND on the *mode* argument and the complement of the current protection mask.
- It defaults to stream format with line-feed record separator and implied carriage-return attributes.

See also `open`, `close`, `read`, `write`, and `lseek` in this section.

## Return Values

|                 |                                                                                                            |
|-----------------|------------------------------------------------------------------------------------------------------------|
| <code>n</code>  | A file descriptor.                                                                                         |
| <code>-1</code> | Indicates errors, including protection violations, undefined directories, and conflicting file attributes. |

## [no]crmode

---

## [no]crmode

In the UNIX system environment, the `crmode` and `nocrmode` functions set and unset the terminal from cbreak mode. In cbreak mode, a single input character can be processed without pressing Return. This mode of single-character input is only supported with the Curses input routine `getch`.

### Format

```
#include <curses.h>
```

```
crmode()
```

```
nocrmode()
```

### Example

```
/* Program to demonstrate the use of crmod() and curses */
#include <curses.h>
main()
{
 WINDOW *win1;
 char vert = '.',
 hor = '.',
 str[80];

 /* Initialize standard screen, turn echo off. */
 initscr();
 noecho();

 /* Define a user window. */
 win1 = newwin(22, 78, 1, 1);

 /* Turn on reverse video and draw a box on border. */
 setattr(_REVERSE);
 box(stdscr, vert, hor);
 mvwaddstr(win1, 2, 2, "Test cbreak input");
 refresh();
 wrefresh(win1);

 /* Set cbreak, do some input, and output it. */
 crmode();
 getch();
 nocrmode(); /* Turn off cbreak. */
 mvwaddstr(win1, 5, 5, str);
 mvwaddstr(win1, 7, 7, "Type something to clear the screen");
 wrefresh(win1);

 /* Get another character, then delete the window. */
 getch();
 wclear(win1);
 touchwin(stdscr);
 endwin();
}
```

In this example, the first call to `getch` returns as soon as one character is entered, because `crmode` was called before `getch` was called. The second time `getch` is called, it waits until the Return key is pressed before processing the character entered, because `nocrmode` was called before `getch` was called the second time.

---

## ctermid

Returns a character string giving the equivalence string of SYS\$COMMAND. This is the name of the controlling terminal.

### Format

```
#include <stdio.h>

char *ctermid (char *str);
```

### Function Variants

The ctermid function has variants named `_ctermid32` and `_ctermid64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Argument

#### **str**

Must be a pointer to an array of characters. If this argument is NULL, the file name is stored internally and might be overwritten by the next ctermid call. Otherwise, the file name is stored beginning at the location indicated by the argument. The argument must point to a storage area of length `L_ctermid` (defined by the `<stdio.h>` header file).

### Return Value

|         |                               |
|---------|-------------------------------|
| pointer | Points to a character string. |
|---------|-------------------------------|

---

## ctime, ctime\_r

Converts a time in seconds, since 00:00:00 January 1, 1970, to an ASCII string in the form generated by the `asctime` function.

### Format

```
#include <time.h>

char *ctime (const time_t *bintim);

char *ctime_r (const time_t *bintim, char *buffer); (ISO POSIX-1)
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to this function that is equivalent to the behavior before OpenVMS Version 7.0.

### Arguments

#### **bintim**

A pointer to a variable that specifies the time value (in seconds) to be converted.

#### **buffer**

A pointer to a character array that is at least 26 bytes long. This array is used to store the generated date-and-time string.

### Description

The `ctime` and `ctime_r` functions convert the time pointed to by `bintim` into a 26-character string, and return a pointer to the string.

The difference between the `ctime_r` and `ctime` functions is that the former puts its result into a user-specified buffer. The latter puts its result into thread-specific static memory allocated by the *Compaq C* RTL, which can be overwritten by subsequent calls to `ctime` or `asctime`; you must make a copy if you want to save it.

On success, `ctime` returns a pointer to the string; `ctime_r` returns its second argument. On failure, these functions return the NULL pointer.

The type `time_t` is defined in the `<time.h>` header file as follows:

```
typedef long int time_t
```

The `ctime` function behaves as if it called `tzset`.

---

#### Note

Generally speaking, UTC-based time functions can affect in-memory time-zone information, which is process-wide data. However, if the system time zone remains the same during the execution of the application (which is the common case) and the cache of timezone files is enabled (which is the default), then the `_r` variant of the time functions `asctime_r`, `ctime_r`, `gmtime_r` and `localtime_r`, is both thread-safe and AST-reentrant.

If, however, the system time zone can change during the execution of the application or the cache of timezone files is not enabled, then both

variants of the UTC-based time functions belong to the third class of functions, which are neither thread-safe nor AST-reentrant.

---

## **Return Values**

|      |                                                            |
|------|------------------------------------------------------------|
| x    | A pointer to the 26-character ASCII string, if successful. |
| NULL | Indicates failure.                                         |

---

**cuserid**

Returns a pointer to a character string containing the name of the user initiating the current process.

**Format**

```
#include <unistd.h> (X/Open, POSIX-1)
```

```
#include <stdio.h> (X/Open)
```

```
char *cuserid (char *str);
```

**Function Variants**

The `cuserid` function has variants named `_cuserid32` and `_cuserid64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

**Argument****`str`**

If this argument is `NULL`, the user name is stored internally. If the argument is not `NULL`, it points to a storage area of length `L_cuserid` (defined by the `<stdio.h>` header file), and the name is written into that storage. If the user name is a null string, the function returns `NULL`.

**Return Values**

pointer

Points to a string.

`NULL`

If the user name is a null string.

---

## DECC\$CRTL\_INIT

Allows you to call the *Compaq C* RTL from other languages or to use the *Compaq C* RTL when your main function is not in C. It initializes the run-time environment and establishes both an exit and condition handler. VAXC\$CRTL\_INIT is a synonym for DECC\$CRTL\_INIT. Either name invokes the same routine.

### Format

```
#include <signal.h>

void DECC$CRTL_INIT(void);
```

### Description

The following example shows a Pascal program that calls the *Compaq C* RTL using the DECC\$CRTL\_INIT function:

```
$ PASCAL EXAMPLE1
$ LINK EXAMPLE1
$ TY EXAMPLE1.PAS
PROGRAM TESTC(input, output);
PROCEDURE DECC$CRTL_INIT; extern;
BEGIN
 DECC$CRTL_INIT;
END
```

A shareable image need only call this function if it contains a *Compaq C* function for signal handling, environment variables, I/O, exit handling, a default file protection mask, or if it is a child process that should inherit context.

Although many of the initialization activities are performed only once, DECC\$CRTL\_INIT can safely be called multiple times. On OpenVMS VAX systems, DECC\$CRTL\_INIT establishes the *Compaq C* RTL internal OpenVMS exception handler in the frame of the routine that calls DECC\$CRTL\_INIT each time DECC\$CRTL\_INIT is called.

At least one frame in the current call stack must have that handler established for OpenVMS exceptions to get mapped to UNIX signals.

## decc\$feature\_get\_index

---

## decc\$feature\_get\_index

Returns an index for accessing feature values.

### Format

```
int decc$feature_get_index (char *name);
```

### Argument

**name**

Pointer to a character string passed as a name in the list of supported features.

### Description

The `decc$feature_get_index` function looks up the string passed as *name* in the list of supported features. If the name is found, `decc$feature_get_index` returns a (nonnegative) index that can be used to set or retrieve the values for the feature. The comparison for *name* is case insensitive.

On error, `-1` is returned and `errno` is set to indicate the error.

See also `decc$feature_get_name`, `decc$feature_get_value`, and `decc$feature_set_value`.

### Return Values

|                 |                                                                                               |
|-----------------|-----------------------------------------------------------------------------------------------|
| <code>n</code>  | A nonnegative index that can be used to set or retrieve the specified values for the feature. |
| <code>-1</code> | Indicates an error; <code>errno</code> is set.                                                |

### Example

The following a sample module sets default values for an application. The module can be compiled separately from the application and included in the image during link:

```
static int set_feature_default(char *name, int value)
{
 int index = decc$feature_get_index(name);
 if (index == -1 ||
 decc$feature_set_value(index, 0, value) == -1)
 {
 perror(name);
 return -1;
 }
 return 0;
}

static void my_init(void)
{
 set_feature_default("DECC$POSIX SEEK STREAM FILE" , TRUE);
 set_feature_default("DECC$ARGV CASE PARSE STYLE" , TRUE);
 set_feature_default("DECC$EFS CASE PRESERVE" , TRUE);
 set_feature_default("DECC$FILE_SHARING" , TRUE);
}
```



```
#pragma nostandard
globaldef { "LIB$INITIALIZ" } readonly _align (LONGWORD)
 int spare[8] = {0};
globaldef { "LIB$INITIALIZE" } readonly _align (LONGWORD)
 void (*x_my_init)() = my_init;
#pragma standard
```

## decc\$feature\_get\_name

---

## decc\$feature\_get\_name

Returns a feature name.

### Format

```
char *decc$feature_get_name (int index);
```

### Argument

**index**

An integer value from 0 to the highest allocated feature.

### Description

The `decc$feature_get_name` function returns a pointer to a null-terminated string containing the name of the feature for the entry specified by *index*. The *index* value can be 0 to the highest allocated feature. If there is no feature corresponding to the *index* value, then the function returns a NULL pointer.

On error, NULL is returned and `errno` is set to indicate the error.

See also `decc$feature_get_index`, `decc$feature_get_value`, and `decc$feature_set_value`.

### Return Values

|      |                                                                                                                  |
|------|------------------------------------------------------------------------------------------------------------------|
| x    | Pointer to a null-terminated string containing the name of the feature for the entry specified by <i>index</i> . |
| NULL | Indicates an error; <code>errno</code> is set.                                                                   |

### Example

See `decc$feature_get_index` for an example of retrieving and setting C RTL features.

---

## decc\$feature\_get\_value

Returns a feature value depending on the *mode* argument.

### Format

```
int decc$feature_get_value (int index, int mode);
```

### Arguments

#### **index**

An integer value from 0 to the highest allocated feature.

#### **mode**

An integer indicating which feature value to return. The values for mode are:

- 0—Default value
- 1—Current value
- 2—Minimum value
- 3—Maximum value
- 4—Initialization state

### Description

The `decc$feature_get_value` function retrieves a value for the feature specified by *index*. The *mode* determines which value is returned.

The default value is what is used if not set by a logical name or overridden by a call to `decc$feature_set_value`.

If *mode* = 4, then the initialization state is returned. Values for the initialization state are:

- 0—not initialized
- 1—set by logical name
- 2—forced by `decc$feature_set_value`
- 1—initialized to default value

On error, –1 is returned and `errno` is set to indicate the error.

See also `decc$feature_get_index`, `decc$feature_get_name`, and `decc$feature_set_value`.

### Return Values

- |          |                                                                                   |
|----------|-----------------------------------------------------------------------------------|
| <i>n</i> | An integer corresponding to the specified <i>index</i> and <i>mode</i> arguments. |
| –1       | Indicates an error; <code>errno</code> is set.                                    |

### Example

See `decc$feature_get_index` for an example of retrieving and setting C RTL features.

## decc\$feature\_set\_value

---

## decc\$feature\_set\_value

Sets the default value or the current value for the feature specified by *index*.

### Format

```
int decc$feature_set_value (int index, int mode, int value);
```

### Arguments

#### **index**

An integer value from 0 to the highest allocated feature.

#### **mode**

An integer indicating whether to set the default or current feature value. The values for mode are:

- 0 - default value
- 1 - current value

#### **value**

The feature value to be set.

### Description

The `decc$feature_set_value` function sets the default value or the current value (as determined by the *mode* argument) for the feature specified by *index*.

If this function is successful, it returns the previous value.

On error, `-1` is returned and `errno` is set to indicate the error.

See also `decc$feature_get_index`, `decc$feature_get_name`, and `decc$feature_get_value`.

### Return Values

- |                 |                                                |
|-----------------|------------------------------------------------|
| <code>n</code>  | The previous feature value.                    |
| <code>-1</code> | Indicates an error; <code>errno</code> is set. |

### Example

See `decc$feature_get_index` for an example of retrieving and setting C RTL features.

---

## decc\$fix\_time

Converts OpenVMS binary system times to UNIX binary times.

### Format

```
#include <unixlib.h>

unsigned int decc$fix_time (void *vms_time);
```

### Argument

#### **vms\_time**

The address of a quadword containing an OpenVMS binary time:

```
unsigned int quadword[2];
unsigned int *vms_time = quadword;
```

### Description

The `decc$fix_time` routine converts an OpenVMS binary system time (a 64-bit quadword containing the number of 100-nanosecond ticks since 00:00 November 17, 1858) to a UNIX binary time (a longword containing the number of seconds since 00:00 January 1, 1970). This routine is useful for converting binary times returned by OpenVMS system services and RMS services to the format used by some *Compaq C* RTL routines, such as `ctime` and `localtime`.

### Return Values

|                    |                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| x                  | A longword containing the number of seconds since 00:00 January 1, 1970.                                                            |
| (unsigned int)(-1) | Indicates an error. Be aware, that a return value of (unsigned int)(-1) can also represent a valid date of Sun Feb 7 06:28:15 2106. |

### Example

```
#include <unixlib.h>
#include <stdio.h>
#include <starlet.h> /* VMS Specific SYS$ routines) */

main()
{
 unsigned int current_vms_time[2]; /* quadword for OpenVMS time */
 unsigned int number_of_seconds; /* number of seconds */

 /* first get the current system time */
 sys$gettim(&current_vms_time[0]);

 /* fix the time */
 number_of_seconds = decc$fix_time(&current_vms_time[0]);

 printf("Number of seconds since 00:00 January 1, 1970 = %d",
 number_of_seconds);
}
```

This example shows how to use the `decc$fix_time` routine in *Compaq C*. It also shows the use of the `SYS$GETTIM` system service.

---

## decc\$from\_vms

Converts OpenVMS file specifications to UNIX style file specifications.

### Format

```
#include <unixlib.h>
```

```
int decc$from_vms (const char *vms_filespec, int action_routine, int wild_flag);
```

### Arguments

#### **vms\_filespec**

The address of a null-terminated string containing a name in OpenVMS file specification format.

#### **action\_routine**

The address of a routine that takes as its only argument a null-terminated string containing the translation of the given OpenVMS file name to a valid UNIX style file name.

If the *action\_routine* returns a nonzero value (TRUE), file translation continues. If it returns a zero value (FALSE), no further file translation takes place.

#### **wild\_flag**

Either 0 or 1, passed by value. If a 0 is specified, wildcards found in *vms\_filespec* are not expanded. Otherwise, wildcards are expanded and each one is passed to *action\_routine*. Only expanded file names that correspond to existing UNIX style files are included.

### Description

The `decc$from_vms` routine converts the given OpenVMS file specification into the equivalent UNIX style file specification. It allows you to specify OpenVMS wildcards, which are translated into a list of corresponding existing files in UNIX style file specification format.

### Return Value

|   |                                                                                     |
|---|-------------------------------------------------------------------------------------|
| x | The number of file names that result from the specified OpenVMS file specification. |
|---|-------------------------------------------------------------------------------------|

### Example

```
/* This example must be run as a foreign command */
/* and be supplied with a VMS file specification */
#include <unixlib.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
 int number_found; /* number of files found */
 int print_name(); /* name printer */

 printf("Translating: %s\n", argv[1]);
 number_found = decc$from_vms(argv[1], print_name, 1);
 printf("\n%d files found", number_found);
}
```

```
/* print the name on each line */
print_name(char *name)
{
 printf("\n%s", name);
 /* will continue as long as success status is returned */
 return (1);
}
```

This example shows how to use the `decc$from_vms` routine in *Compaq C*. It produces a simple form of the `ls` command that lists existing files that match an OpenVMS file specification supplied on the command line. The matching files are displayed in UNIX style file specification format.

## decc\$match\_wild

---

## decc\$match\_wild

Matches a string to a pattern.

### Format

```
#include <unixlib.h>

int decc$match_wild (char *test_string, char *string_pattern);
```

### Arguments

#### **test\_string**

The address of a null-terminated string.

#### **string\_pattern**

The address of a string containing the pattern to be matched. This pattern can contain wildcards (such as asterisks (\*), question marks (?), and percent signs (%)) as well as regular expressions (such as the range [a-z]).

### Description

The `decc$match_wild` routine determines whether the specified test string is a member of the set of strings specified by the pattern.

### Return Values

|           |                                        |
|-----------|----------------------------------------|
| 1 (TRUE)  | The string matches the pattern.        |
| 0 (FALSE) | The string does not match the pattern. |

### Example

```
/* Define as a foreign command and then provide two arguments */
/* match_wild string_to_test pattern */
#include <unixlib.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
 if (decc$match_wild(argv[1], argv[2]))
 printf("\n%s matches %s", argv[1], argv[2]);
 else
 printf("\n%s does not match %s", argv[1], argv[2]);
}
```



---

## decc\$record\_read

Reads a record from a file.

### Format

```
#include <stdio.h>

int decc$record_read (FILE *fp, void *buffer, int nbytes);
```

### Arguments

**fp**

A file pointer. The specified file pointer must refer to a file currently opened for reading.

**buffer**

The address of contiguous storage in which the input data is placed.

**nbytes**

The maximum number of bytes involved in the read operation.

### Description

The decc\$record\_read function is specific to OpenVMS systems and should not be used when writing portable applications.

This function is equivalent to the read function, except that the first argument is a file pointer, not a file descriptor.

### Return Values

|    |                                                                                                                                                     |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | The number of characters read.                                                                                                                      |
| -1 | Indicates a read error, including physical input errors, illegal buffer addresses, protection violations, undefined file descriptors, and so forth. |

## decc\$record\_write

---

## decc\$record\_write

Writes a record to a file.

### Format

```
#include <stdio.h>

int decc$record_write (FILE *fp, void *buffer, int nbytes);
```

### Arguments

**fp**

A file pointer. The specified file pointer must refer to a file currently opened for writing or updating.

**buffer**

The address of contiguous storage from which the output data is taken.

**nbytes**

The maximum number of bytes involved in the write operation.

### Description

The `decc$record_write` function is specific to OpenVMS systems and should not be used when writing portable applications.

This function is equivalent to the `write` function, except that the first argument is a file pointer, not a file descriptor.

### Return Values

x

The number of bytes written.

−1

Indicates errors, including undefined file descriptors, illegal buffer addresses, and physical I/O errors.

---

## decc\$set\_child\_standard\_streams

For a child spawned by a function from the exec family of functions, associates specified file descriptors with a child's standard streams: stdin, stdout, and stderr.

### Format

```
#include <unistd.h>

int decc$set_child_standard_streams (int fd1, int fd2, int fd3);
```

### Arguments

#### fd1

The file associated with this file descriptor in the parent process is associated with file descriptor number 0 (stdin) in the child process. If -1 is specified, the file associated with the parent's file descriptor number 0 is used (the default).

#### fd2

The file associated with this file descriptor in the parent process is associated with file descriptor number 1 (stdout) in the child process. If -1 is specified, the file associated with the parent's file descriptor number 1 is used (the default).

#### fd3

The file associated with this file descriptor in the parent process is associated with file descriptor number 2 (stderr) in the child process. If -1 is specified, the file associated with the parent's file descriptor number 2 is used (the default).

### Description

The decc\$set\_child\_standard\_streams function allows mapping of specified file descriptors to the child's stdin/stdout/stderr streams, thereby compensating, to a certain degree, the lack of a real fork function on OpenVMS systems.

On UNIX systems, the code between fork and exec is executed in the context of the child process:

```
parent:
 create pipes p1, p2 and p3
 fork
child:
 map stdin to p1 like dup2(stdin, p1);
 map stdout to p2 like dup2(stdout, p2);
 map stderr to p3 like dup2(stderr, p3);
 exec (child reads from stdin and writes to stdout and stderr)
 exit
parent:
 communicates with the child using pipes
```

On OpenVMS systems, the same task could be achieved as follows:

```
parent:
 create pipes p1, p2 and p3
 decc$set_child_standard_streams(p1, p2, p3);
 vfork
 exec (child reads from stdin and writes to stdout and stderr)
parent:
 communicates with the child using pipes
```

## decc\$set\_child\_standard\_streams

Once established through the call to `decc$set_child_standard_streams`, the mapping of the child's standard streams remains in effect until explicitly disabled by one of the following calls:

```
decc$set_child_standard_streams(-1, -1, -1);
```

OR

```
decc$set_child_standard_streams(0, 1, 2);
```

Usually, the child process inherits all its parent's open file descriptors. However, if file descriptor number *n* was specified in the call to `decc$set_child_standard_streams`, it is not inherited by the child process as file descriptor number *n*; instead, it becomes one of the child's standard streams.

---

### Notes

---

- Standard streams can be redirected only to pipes.
- If the parent process redefines the DCL DEFINE command, this redefinition is not in effect in a subprocess with user-defined channels. The subprocess always sees the standard DCL DEFINE command.
- It is the responsibility of the parent process to consume all the output written by the child process to `stdout` and `stderr`. Depending on how the subprocess writes to `stdout` and `stderr`—in wait or nowait mode – the subprocess might be placed in LEF state waiting for the reader. For example, DCL writes to `SYS$OUTPUT` and `SYS$ERROR` in a wait mode, so a child process executing a DCL command procedure will wait until all the output is read by the parent process.

Recommendation: Read the pipes associated with the child process' `stdout` and `stderr` in a loop until an EOF message is received, or declare write attention ASTs on these mailboxes.

- The amount of data written to `SYS$OUTPUT` depends on the verification status of the process (`SET VERIFY/NOVERIFY` command); the subprocess inherits the verification status of the parent process. It is the caller's responsibility to set the verification status of the parent process to match the expected amount of data written to `SYS$OUTPUT` by the subprocess.
- Some applications, like DTM, define `SYS$ERROR` as `SYS$OUTPUT`. If `stderr` is not redefined by the caller, it is set in the subprocess as the parent's `SYS$ERROR`, which in this case translates to the parent's `SYS$OUTPUT`.

If the caller redefines `stdout` to a pipe and does not redefine `stderr`, output sent to `stderr` goes to the pipe associated with `stdout`, and the amount of data written to this mailbox may be more than expected. Although redefinition of any subset of standard channels is supported, it is always safe to explicitly redefine all of them (or at least `stdout` and `stderr`) to avoid this situation.

- For a child process executing a DCL command procedure, `SYS$COMMAND` is set to the pipe specified for the child's `stdin` so that the parent process can feed the child requesting data from `SYS$COMMAND` through the pipe. For DCL command procedures, it is impossible to pass data from the parent to the child by means of the

child's SYS\$INPUT because for a command procedure, DCL defines SYS\$INPUT as the command file itself.

---

## Return Values

|    |                                                                                                                              |
|----|------------------------------------------------------------------------------------------------------------------------------|
| x  | The number of file descriptors set for the child. This number does not include file descriptors specified as -1 in the call. |
| -1 | indicates that an invalid file descriptor was specified; errno is set to EBADF.                                              |

## Example

```
parent.c
=====
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int decc$set_child_standard_streams(int, int, int);

main()
{
 int fdin[2], fdout[2], fderr[2];
 char msg[] = "parent writing to child's stdin";
 char buf[80];
 int nbytes;

 pipe(fdin);
 pipe(fdout);
 pipe(fderr);

 if (vfork() == 0) {
 decc$set_child_standard_streams(fdin[0], fdout[1], fderr[1]);
 execl("child", "child");
 }
 else {
 write(fdin[1], msg, sizeof(msg));
 nbytes = read(fdout[0], buf, sizeof(buf));
 buf[nbytes] = '\0';
 puts(buf);
 nbytes = read(fderr[0], buf, sizeof(buf));
 buf[nbytes] = '\0';
 puts(buf);
 }
}

child.c
=====
#include <stdio.h>
#include <unistd.h>

main()
{
 char msg[] = "child writing to stderr";
 char buf[80];
 int nbytes;

 nbytes = read(0, buf, sizeof(buf));
 write(1, buf, nbytes);
 write(2, msg, sizeof(msg));
}
```

## decc\$set\_child\_standard\_streams

```
child.com
=====
$ read sys$command s
$ write sys$output s
$ write sys$error "child writing to stderr"
```

This example program returns the following for both `child.c` and `child.com`:

```
$ run parent
parent writing to child's stdin
child writing to stderr
```

Note that in order to activate `child.com`, you must explicitly specify `execl("child.com", ...)` in the `parent.c` program.

---

## decc\$set\_reentrancy

Controls the type of reentrancy that reentrant *Compaq C* RTL routines will exhibit.

### Format

```
#include <reentrancy.h>

int decc$set_reentrancy (int type);
```

### Argument

#### type

The type of reentrancy desired. Use one of the following values:

- **C\$C\_MULTITHREAD** — Designed to be used in conjunction with the DECthreads product. It performs DECthreads locking and never disables ASTs. DECthreads must be available on your system to use this form of reentrancy.
- **C\$C\_AST** — Uses the `_BBSSI` (*VAX only*) or `__TESTBITSSI` (*Alpha only*) built-in function to perform simple locking around critical sections of RTL code, and it may additionally disable asynchronous system traps (ASTs) in locked regions of code. This type of locking should be used when AST code contains calls to *Compaq C* RTL I/O routines, or when the user application disables ASTs.
- **C\$C\_TOLERANT** — Uses the `_BBSSI` (*VAX only*) or `__TESTBITSSI` (*Alpha only*) built-in function to perform simple locking around critical sections of RTL code, but ASTs are not disabled. This type of locking should be used when ASTs are used and must be delivered immediately. TOLERANT is the default reentrancy type.
- **C\$C\_NONE** — Gives optimal performance in the *Compaq C* RTL, but does absolutely no locking around critical sections of RTL code. It should only be used in a single-threaded environment when there is no chance that the thread of execution will be interrupted by an AST that would call the *Compaq C* RTL.

The reentrancy type can be raised but never lowered. The ordering of reentrancy types from low to high is C\$C\_NONE, C\$C\_TOLERANT, C\$C\_AST and C\$C\_MULTITHREAD. For example, once an application is set to multithread, a call to set the reentrancy to AST is ignored. A call to `decc$set_reentrancy` that attempts to lower the reentrancy type returns a value of `-1`.

### Description

Use the `decc$set_reentrancy` function to change the type of reentrancy exhibited by reentrant routines.

`decc$set_reentrancy` must be called exclusively at the non-AST level.

In an application using DECthreads, DECthreads automatically sets the reentrancy to multithread.

## **decc\$set\_reentrancy**

### **Return Value**

type  
–1

The type of reentrancy used before this call.  
The reentrancy was set to a lower type.



---

## decc\$to\_vms

Converts UNIX style file specifications to OpenVMS file specifications.

### Format

```
#include <unixlib.h>
```

```
int decc$to_vms (const char *unix_style_filespec, int (*action_routine)(char *unix_style_filespec, int
type_of_file), int allow_wild, int no_directory);
```

### Arguments

#### unix\_style\_filespec

The address of a null-terminated string containing a name in UNIX style file specification format.

#### action\_routine

The address of a routine that accepts the following arguments:

- A pointer to a null-terminated string that contains the UNIX style file name to be translated to a valid OpenVMS file name
- An integer that has one of the following values:

| Value                 | Translation                                                                           |
|-----------------------|---------------------------------------------------------------------------------------|
| 0 (DECC\$K_FOREIGN)   | A file on a remote system that is not running the OpenVMS or VAXELN operating system. |
| 2 (DECC\$K_DIRECTORY) | The OpenVMS translation of the UNIX style file name is a directory.                   |
| 1 (DECC\$K_FILE)      | The translation is a file.                                                            |

These values can be defined symbolically with the symbols DECC\$K\_FOREIGN, DECC\$K\_DIRECTORY, and DECC\$K\_FILE. See the example for more information.

If *action\_routine* returns a nonzero value (TRUE), file translation continues. If it returns a 0 value (FALSE), no further file translation takes place.

#### allow\_wild

Either 0 or 1, passed by value. If a 0 is specified, wildcards found in *unix\_style\_filespec* are not expanded. Otherwise, wildcards are expanded and each one is passed to *action\_routine*. Only expanded file names that correspond to existing OpenVMS files are included.

#### no\_directory

An integer that has one of the following values:

| Value | Translation                                          |
|-------|------------------------------------------------------|
| 0     | Directory is not allowed.                            |
| 1     | Prevent expansion of the string as a directory name. |
| 2     | Forced to be a directory name.                       |

## decc\$to\_vms

### Description

The `decc$to_vms` routine converts the given UNIX style file specification into the equivalent OpenVMS file specification (in all uppercase letters). It allows you to specify UNIX style wildcards, which are translated into a list of corresponding OpenVMS files.

### Return Value

|   |                                                                                        |
|---|----------------------------------------------------------------------------------------|
| x | The number of file names that result from the specified UNIX style file specification. |
|---|----------------------------------------------------------------------------------------|

### Example

```
/* Translate "UNIX" wildcard file names to VMS names */
/* Define as a foreign command and provide the name */
/* as an argument. */
#include <unixlib.h>
#include <stdio.h>
int print_name(char *, int);
int main(int argc, char *argv[])
{
 int number_found; /* number of files found */
 printf("Translating: %s\n", argv[1]);
 number_found = decc$to_vms(argv[1], print_name, 1, 0);
 printf("%d files found\n", number_found);
}
/* action routine that prints name and type on each line */
int print_name(char *name, int type)
{
 if (type == DECC$K_DIRECTORY)
 printf("directory: %s\n", name);
 else if (type == DECC$K_FOREIGN)
 printf("remote non-VMS: %s\n", name);
 else
 printf("file: %s\n", name);
 /* Translation continues as long as success status is returned */
 return (1);
}
```

This example shows how to use the `decc$to_vms` routine in *Compaq C*. It takes a UNIX style file specification argument and displays, in OpenVMS file specification format, the name of each existing file that matches it.

---

## decc\$translate\_vms

Translates OpenVMS file specifications to UNIX style file specifications.

### Format

```
#include <unixlib.h>

char *decc$translate_vms (const char *vms_filespec);
```

### Argument

#### **vms\_filespec**

The address of a null-terminated string containing a name in OpenVMS file specification format.

### Description

The `decc$translate_vms` function translates the given OpenVMS file specification into the equivalent UNIX style file specification, whether or not the file exists. The translated name string is stored in a thread-specific memory, which is overwritten by each call to `decc$translate_vms` from the same thread.

This function differs from the `decc$from_vms` function, which does the conversion for existing files only.

### Return Values

|    |                                                                                                                                                                                                                  |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | The address of a null-terminated string containing a name in UNIX style file specification format.                                                                                                               |
| 0  | Indicates that the file name is null or syntactically incorrect.                                                                                                                                                 |
| -1 | Indicates that the file specification contains an ellipsis (for example, [ . . . ]a.dat), but is otherwise correct. You cannot translate the OpenVMS ellipsis syntax into a valid UNIX style file specification. |

### Example

```
/* Demonstrate translation of a "UNIX" name to VMS form */
/* define a foreign command and pass the name as the */
/* argument. */
#include <unixlib.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
 char *ptr; /* translation result */
 ptr = decc$translate_vms(argv[1]);
 if ((int) ptr == 0 || (int) ptr == -1)
```

## decc\$translate\_vms

```
 printf("could not translate %s\n", argv[1]);
 else
 printf("%s is translated to %s\n", argv[1], ptr);
}
```

---

## decc\$validate\_wchar

Confirms that its argument is a valid wide character in the current program's locale.

### Format

```
#include <unistd.h>

int decc$validate_wchar (wchar_t wc);
```

### Argument

**wc**  
Wide character to be validated.

### Description

The `decc$validate_wchar` function provides a convenient way to verify whether a specified argument of `wchar_t` type is a valid wide character in the current program's locale.

One reason to call `decc$validate_wchar` is that the `isw*` wide-character classification functions and macros do not validate their argument before dereferencing the `classmask` array describing character properties. Passing an `isw*` function a value that exceeds the maximum wide-character value for the current program's locale can result in an attempt to access memory beyond the allocated `classmask` array.

A standard way to validate a wide character is to call the `wctomb` function, but this way is less convenient because it requires declaring a multibyte character array of sufficient size and passing it to `wctomb`.

### Return Values

|   |                                                                                                                                           |
|---|-------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Indicates that the specified wide character is a valid wide character in the current program's locale.                                    |
| 0 | Indicates that the specified wide character is not a valid wide character in the current program's locale. <code>errno</code> is not set. |

## decc\$write\_eof\_to\_mbx

---

## decc\$write\_eof\_to\_mbx

Writes an end-of-file message to the mailbox.

### Format

```
#include <unistd.h>

int decc$write_eof_to_mbx (int fd);
```

### Argument

**fd**  
File descriptor associated with the mailbox.

### Description

The `decc$write_eof_to_mbx` function writes end-of-file message to the mailbox.

For a mailbox that is not a pipe, the write function called with an *nbytes* argument value of 0 sends an end-of-file message to the mailbox. For a pipe, however, the only way to write an end-of-file message to the mailbox is to close the pipe.

If the child's standard input is redirected to a pipe through a call to the `decc$set_child_standard_streams` function, the parent process can call `decc$write_eof_to_mbx` for this pipe to send an EOF message to the child. It has the same effect as if the child read the data from a terminal, and Ctrl/Z was pressed.

After a call to `decc$write_eof_to_mbx`, the pipe can be reused for communication with another child, for example. This is the purpose of `decc$write_eof_to_mbx`: to allow reuse of the pipe instead of having to close it just to send an end-of-file message.

### Return Values

|    |                                                                                                                                                 |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success                                                                                                                               |
| -1 | Indicates failure; <code>errno</code> and <code>vaxc\$errno</code> are set according to the failure status returned by <code>SYS\$QIOW</code> . |

### Example

```
/* decc$write_eof_to_mbx_example.c */
#include <errno.h>
#include <stdio.h>
#include <string.h>

#include <fcntl.h>
#include <unistd.h>
#include <unixio.h>

#include <descrip.h>
#include <ssdef.h>
#include <starlet.h>

int decc$write_eof_to_mbx(int);
```

```

main()
{
 int status, nbytes, failed = 0;
 int fd, fd2[2];
 short int channel;
 $DESCRIPTOR(mbxname_dsc, "TEST_MBX");
 char c;

 /* first try a mailbox created by SYS$CREMBX */
 status = sys$crembx(0, &channel, 0, 0, 0, 0, &mbxname_dsc, 0, 0);
 if (status != SS$ NORMAL) {
 printf("sys$crembx failed: %s\n",strerror(EVMSERR, status));
 failed = 1;
 }

 if ((fd = open(mbxname_dsc.dsc$a_pointer, O_RDWR, 0)) == -1) {
 perror("? open mailbox");
 failed = 1;
 }

 if (decc$write_eof_to_mbx(fd) == -1) {
 perror("? decc$write_eof_to_mbx to mailbox");
 failed = 1;
 }

 if ((nbytes = read(fd, &c, 1)) != 0 || errno != 0) {
 perror("? read mailbox");
 printf("? nbytes = %d\n", nbytes);
 failed = 1;
 }

 if (close(fd) == -1) {
 perror("? close mailbox");
 failed = 1;
 }

 /* Now do the same thing with a pipe */
 errno = 0; /* Clear errno for consistency */
 if (pipe(fd2) == -1) {
 perror("? opening pipe");
 failed = 1;
 }

 if (decc$write_eof_to_mbx(fd2[1]) == -1) {
 perror("? decc$write_eof_to_mbx to pipe");
 failed = 1;
 }

 if ((nbytes = read(fd2[0], &c, 1)) != 0 || errno != 0) {
 perror("? read pipe");
 printf("? nbytes = %d\n", nbytes);
 failed = 1;
 }

 /* Close both file descriptors involved with the pipe */
 if (close(fd2[0]) == -1) {
 perror("close(fd2[0])");
 failed = 1;
 }

 if (close(fd2[1]) == -1) {
 perror("close(fd2[1])");
 failed = 1;
 }
}

```

## decc\$write\_eof\_to\_mbx

```
if (failed)
 puts("?Example program failed");
else
 puts("Example ran to completion");
}
```

This example program produces the following result:

Example ran to completion



---

**[w]delch**

Delete the character on the specified window at the current position of the cursor. The `delch` function operates on the `stdscr` window.

**Format**

```
#include <curses.h>
int delch();
int wdelch (WINDOW *win);
```

**Argument**

**win**  
A pointer to the window.

**Description**

All of the characters to the right of the cursor on the same line are shifted to the left, and a blank character is appended to the end of the line.

**Return Values**

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

## delete

---

## delete

Deletes a file.

### Format

```
#include <unixio.h>

int delete (const char *file_spec);
```

### Argument

#### **file\_spec**

A pointer to the string that is an OpenVMS or UNIX style file specification. The file specification can include a wildcard in its version number (but not in any other part of the file spec). So, for example, files of the form *filename.txt*;\* can be deleted.

### Description

If you specify a directory in the file name and it is a search list that contains an error, *Compaq C* for *OpenVMS* Systems interprets it as a file error.

The remove and delete functions are functionally equivalent in the *Compaq C* RTL.

See also remove in this section.

---

#### **Note**

The delete routine is not available to C++ programmers because it conflicts with the C++ reserved word delete. C++ programmers should use the ANSI/ISO C standard function remove instead.

---

### Return Values

|               |                                          |
|---------------|------------------------------------------|
| 0             | Indicates success.                       |
| nonzero value | Indicates that the operation has failed. |

---

**[w]deleteln**

Delete the line at the current position of the cursor. The `deleteln` function acts on the `stdscr` window.

**Format**

```
#include <curses.h>

int deleteln();

int wdeleteln (WINDOW *win);
```

**Argument**

**win**  
A pointer to the window.

**Description**

Every line below the deleted line moves up, and the bottom line becomes blank. The current  $(y,x)$  coordinates of the cursor remain unchanged.

**Return Values**

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

## delwin

---

## delwin

Deletes the specified window from memory.

### Format

```
#include <curses.h>
int delwin (WINDOW *win);
```

### Argument

**win**  
A pointer to the window.

### Description

If the window being deleted contains a subwindow, the subwindow is invalidated. Delete subwindows before deleting their parent. The `delwin` function refreshes all windows covered by the deleted window.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

---

## difftime

Computes the difference, in seconds, between the two times specified by the *time1* and *time2* arguments.

### Format

```
#include <time.h>

double difftime (time_t time2, time_t time1);
```

### Arguments

**time2**  
A time value of type `time_t`.

**time1**  
A time value of type `time_t`.

### Description

The type `time_t` is defined in the `<time.h>` header file as follows:

```
typedef unsigned long int time_t
```

### Return Value

n *time2* − *time1* in seconds expressed as a double.

## dirname

---

## dirname

Reports the parent directory name of a file pathname.

### Format

```
#include <libgen.h>

char *dirname (char *path);
```

### Function Variants

The `dirname` function has variants named `_dirname32` and `_dirname64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Argument

**path**  
The file pathname.

### Description

The `dirname` function takes a pointer to a character string that contains a UNIX path name and returns a pointer to a string that is a pathname of the parent directory of that file. Trailing `'/'` (slash) characters in the path are not counted as part of the path.

This function returns a pointer to the string `"."` (dot), when the *path* argument:

- Does not contain a `'/'` (slash).
- Is a NULL pointer.
- Points to an empty string.

The `dirname` function can modify the string pointed to by the *path* argument.

The `dirname` and `basename` functions together yield a complete pathname. The expression `dirname(path)` obtains the pathname of the directory where `basename(path)` is found.

See also `basename` in this section.

### Return Values

|     |                                                                                                                                                                                            |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x   | A pointer to a string that is the parent directory of the <i>path</i> argument.                                                                                                            |
| "." | The <i>path</i> argument: <ul style="list-style-type: none"><li>• Does not contain a <code>'/'</code> (slash).</li><li>• Is a NULL pointer.</li><li>• Points to an empty string.</li></ul> |

**Example**

Using the `dirname` function, the following example reads a pathname, changes the current working directory to the parent directory, and opens a file.

```
char path [MAXPATHLEN], *pathcopy;
int fd;

fgets(path, MAXPATHLEN, stdin);
pathcopy = strdup(path);
chdir(dirname(pathcopy));
fd = open(basename(path), O_RDONLY);
```

## div

---

## div

Returns the quotient and the remainder after the division of its arguments.

### Format

```
#include <stdlib.h>

div_t div (int numer, int denom);
```

### Arguments

**numer**

A numerator of type int.

**denom**

A denominator of type int.

### Description

The type `div_t` is defined in the standard header file `<stdlib.h>` as follows:

```
typedef struct
{
 int quot, rem;
} div_t;
```



---

## dlclose

Deallocates the address space for a shared library.

### Format

```
#include <dlfcn.h>

void dlclos (void *handle);
```

### Argument

**handle**  
Pointer to the shared library.

### Description

The dlclos function deallocates the address space allocated by the *Compaq C* RTL for the handle.

There is no way on OpenVMS systems to "unload" a shareable image dynamically loaded by the LIB\$FIND\_IMAGE\_SYMBOL routine, which is the routine called by the dlsym function. In other words, there is no way on OpenVMS systems to release the address space occupied by the shareable image brought into memory by dlsym.

## dlerror

---

## dlerror

Returns a string describing the last error that occurred from a call to `dlopen`, `dlclose`, or `dlsym`.

### Format

```
#include <dlfcn.h>

char *dlerror (void);
```

### Return Value

|   |                                                                                                                                      |
|---|--------------------------------------------------------------------------------------------------------------------------------------|
| x | A string describing the last error that occurred from a call to <code>dlopen</code> , <code>dlclose</code> , or <code>dlsym</code> . |
|---|--------------------------------------------------------------------------------------------------------------------------------------|

---

## dlopen

Provides an interface to the dynamic library loader to allow shareable images to be loaded and called at run time.

### Format

```
#include <dlfcn.h>

void *dlopen (char *pathname, int mode);
```

### Arguments

**pathname**

The name of the shareable image. This name is saved for subsequent use by the dlsym function.

**mode**

This argument is ignored on OpenVMS systems.

### Description

The dlopen function provides an interface to the dynamic library loader to allow shareable images to be loaded and called at run time.

This function does not load a shareable image but rather saves its *pathname* argument for subsequent use by the dlsym function. dlsym is the function that actually loads the shareable image through a call to LIB\$FIND\_IMAGE\_SYMBOL.

The *pathname* argument of the dlopen function must be the name of the shareable image. This name is passed as-is by the dlsym function to the LIB\$FIND\_IMAGE\_SYMBOL routine as the *filename* argument. No *image-name* argument is specified in the call to LIB\$FIND\_IMAGE\_SYMBOL, so default file specification of SYS\$SHARE:.EXE is applied to the image name.

The dlopen function returns a handle that is used by a dlsym or dlclose call. If an error occurs, a NULL pointer is returned.

### Return Values

|      |                                                 |
|------|-------------------------------------------------|
| x    | A handle to be used by a dlsym or dlclose call. |
| NULL | Indicates an error.                             |

## dlsym

---

## dlsym

Returns the address of the symbol name found in a shareable image.

### Format

```
#include <dlfcn.h>

void *dlsym (void *handle, char *name);
```

### Arguments

**handle**  
Pointer to the shareable image.

**name**  
Pointer to the symbol name.

### Description

The `dlsym` function returns the address of the symbol name found in the shareable image corresponding to *handle*. If the symbol is not found, a `NULL` pointer is returned.

### Return Values

|      |                                          |
|------|------------------------------------------|
| x    | Address of the symbol name found.        |
| NULL | Indicates that the symbol was not found. |

---

## drand48

Generates uniformly distributed pseudorandom-number sequences. Returns 48-bit, nonnegative, double-precision floating-point values.

### Format

```
#include <stdlib.h>

double drand48 (void);
```

### Description

The `drand48` function generates pseudorandom numbers using the linear congruential algorithm and 48-bit integer arithmetic.

It returns nonnegative, double-precision, floating-point values uniformly distributed over the range of  $y$  values such that  $0.0 \leq y < 1.0$ .

Before you call `drand48`, use either `srand48`, `seed48`, or `lcong48` to initialize the random-number generator. You must initialize prior to invoking the `drand48` function because it stores the last 48-bit  $X_i$  generated into an internal buffer. (Although it is not recommended, constant default initializer values are supplied automatically if the `drand48`, `lrand48`, or `mrnd48` functions are called without first calling an initialization function.)

The `drand48` function works by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

The argument  $m$  equals  $2^{48}$ , so 48-bit integer arithmetic is performed. Unless you invoke `lcong48`, the multiplier value  $a$  and the addend value  $c$  are:

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

The values returned by `drand48` are computed by first generating the next 48-bit  $X_i$  in the sequence. Then the appropriate bits, according to the type of returned data item, are copied from the high-order (most significant) bits of  $X_i$  and transformed into the returned value.

See also `srand48`, `seed48`, `lcong48`, `lrand48`, and `mrnd48` in this section.

### Return Value

|   |                                                        |
|---|--------------------------------------------------------|
| n | A nonnegative, double-precision, floating-point value. |
|---|--------------------------------------------------------|

## dup, dup2

---

## dup, dup2

Allocate a new descriptor that refers to a file specified by a file descriptor returned by `open`, `creat`, or `pipe`.

### Format

```
#include <unistd.h>

int dup (int file_desc1);
int dup2 (int file_desc1, int file_desc2);
```

### Arguments

**`file_desc1`**

The file descriptor being duplicated.

**`file_desc2`**

The new file descriptor to be assigned to the file designated by *file\_desc1*.

### Description

The `dup` function causes a previously unallocated descriptor to refer to its argument, while the `dup2` function causes its second argument to refer to the same file as its first argument.

The argument *file\_desc1* is invalid if it does not describe an open file; *file\_desc2* is invalid if the new file descriptor cannot be allocated. If *file\_desc2* is connected to an open file, that file is closed.

### Return Values

|                 |                                                                |
|-----------------|----------------------------------------------------------------|
| <code>n</code>  | The new file descriptor.                                       |
| <code>-1</code> | Indicates that an invalid argument was passed to the function. |

---

**[no]echo**

Set the terminal so that characters may or may not be echoed on the terminal screen. This mode of single-character input is only supported with Curses.

**Format**

```
#include <curses.h>

void echo (void);
void noecho (void);
```

**Description**

The noecho function may be helpful when accepting input from the terminal screen with wgetch and wgetstr; it prevents the input characters from being written onto the screen.

---

## ecvt

Converts its argument to a null-terminated string of ASCII digits and returns the address of the string. The string is stored in a thread-specific memory location created by the *Compaq C* RTL.

### Format

```
#include <stdlib.h>

char *ecvt (double value, int ndigits, int *decpt, int *sign);
```

### Arguments

**value**

An object of type double that is converted to a null-terminated string of ASCII digits.

**ndigits**

The number of ASCII digits to be used in the converted string.

**decpt**

The position of the decimal point relative to the first character in the returned string. A negative int value means that the decimal point is *decpt* number of spaces to the left of the returned digits, (the spaces being filled with zeros). A 0 value means that the decimal point is immediately to the left of the first digit in the returned string.

**sign**

An integer value that indicates whether the *value* argument is positive or negative. If *value* is negative, the function places a nonzero value at the address specified by *sign*. Otherwise, the function assigns 0 to the address specified by *sign*.

### Description

The *ecvt* function converts *value* to a null-terminated string of length *ndigits*, and returns a pointer to it. The resulting low-order digit is rounded to the correct digit for outputting *ndigits* digits in C E-format. The *decpt* argument is assigned the position of the decimal point relative to the first character in the string.

Repeated calls to the *ecvt* function overwrite any existing string.

The *ecvt*, *fcvt*, and *gcvt* functions represent the following special values specified in the IEEE Standard for floating-point arithmetic:

| Value          | Representation |
|----------------|----------------|
| Quiet NaN      | NaNQ           |
| Signalling NaN | NaNS           |
| +Infinity      | Infinity       |
| –Infinity      | –Infinity      |

The sign associated with each of these values is stored into the *sign* argument. In IEEE floating-point representation, a value of 0 (zero) can be positive or negative, as set by the *sign* argument.



See also `gcvt` and `fcvt` in this section.

## **Return Value**

|                |                                    |
|----------------|------------------------------------|
| <code>x</code> | The value of the converted string. |
|----------------|------------------------------------|

## endpwent

---

## endpwent

Closes the user database.

### Format

```
#include <stdio.h>
#include <pwd.h>
void endpwent (void);
```

### Description

The `endpwent` function closes the user database. The user database basic user attributes are accessed with the `getpwent`, `getpwuid`, `getpwnam`, or `setpwent` functions.

See also `getpwent`, `getpwuid`, `getpwnam`, and `setpwent` in this section

---

**endwin**

Clears the terminal screen and frees any virtual memory allocated to Curses data structures.

**Format**

```
#include <curses.h>

void endwin (void);
```

**Description**

A program that calls Curses functions must call the `endwin` function before exiting to restore the previous environment of the terminal screen.

---

## erand48

Generate uniformly distributed pseudorandom-number sequences. Returns 48-bit nonnegative, double-precision, floating-point values.

### Format

```
#include <stdlib.h>

double erand48 (unsigned short int xsubi[3]);
```

### Argument

#### **xsubi**

An array of three short int, which form a 48-bit integer when concatenated together.

### Description

The `erand48` function generates pseudorandom numbers using the linear congruential algorithm and 48-bit integer arithmetic.

It returns nonnegative, double-precision, floating-point values uniformly distributed over the range of  $y$  values, such that,  $0.0 \leq y < 1.0$ .

The `erand48` function works by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

The argument  $m$  equals  $2^{48}$ , so 48-bit integer arithmetic is performed. Unless you invoke the `lcong48` function, the multiplier value  $a$  and the addend value  $c$  are:

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

The `erand48` function requires that the calling program pass an array as the `xsubi` argument. For the first call, the array must be initialized to the value of the pseudorandom-number sequence. Unlike the `drand48` function, it is not necessary to call an initialization function prior to the first call.

By using different arguments, the `erand48` function allows separate modules of a large program to generate several independent sequences of pseudorandom numbers; for example, the sequence of numbers that one module generates does not depend upon how many times the function is called by other modules.

### Return Values

|                |                                                        |
|----------------|--------------------------------------------------------|
| <code>n</code> | A nonnegative, double-precision, floating-point value. |
|----------------|--------------------------------------------------------|

---

**[w]erase**

Erase the window by painting it with blanks. The erase function acts on the stdscr window.

**Format**

```
#include <curses.h>
int erase();
int werase (WINDOW *win);
```

**Argument**

**win**  
A pointer to the window.

**Description**

Both the erase and werase functions leave the cursor at the current position on the terminal screen after completion; they do not return the cursor to the home coordinates of (0,0).

**Return Values**

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

## erf

---

## erf

Returns the error function of its argument.

### Format

```
#include <math.h>

double erf (double x);
float erff (float x); (Alpha only)
long double erfl (long double x); (Alpha only)
double erfc (double x); (Alpha only)
float erfcf (float x); (Alpha only)
long double erfc1 (long double x); (Alpha only)
```

### Argument

**x**  
A radian expressed as a real number.

### Description

The erf functions return the error function of **x**, where  $\text{erf}(x)$ ,  $\text{erff}(x)$ , and  $\text{erfl}(x)$  equal  $2/\sqrt{\pi}$  times the area under the curve  $e^{-(t^2)}$  between 0 and  $x$ .

The erfc functions return  $(1.0 - \text{erf}(x))$ . The erfc function can result in an underflow as  $x$  gets large.

### Return Values

|            |                                                                               |
|------------|-------------------------------------------------------------------------------|
| <b>x</b>   | The value of the error function (erf) or complementary error function (erfc). |
| <b>NaN</b> | $x$ is NaN; errno is set to EDOM.                                             |
| <b>0</b>   | Underflow occurred; errno is set to ERANGE.                                   |

---

## exec1

Passes the name of an image to be activated in a child process. This function is nonreentrant.

### Format

```
#include <unistd.h>

int exec1 (const char *file_spec, const char *arg0, . . . , (char *)0); (ISO POSIX-1)

int exec1 (char *file_spec, . . .); (Compatibility)
```

### Arguments

#### **file\_spec**

The full file specification of a new image to be activated in the child process.

#### **arg0, ...**

A sequence of pointers to null-terminated character strings.

If the POSIX-1 format is used, at least one argument must be present and must point to a string that is the same as the new process file name (or its last component). (This pointer can also be the NULL pointer, but then `exec1` would accomplish nothing.) The last pointer must be the NULL pointer. This is also the convention if the compatibility format is used.

### Description

To understand how the `exec` functions operate, consider how the OpenVMS system calls any *Compaq C* program, as shown in the following syntax:

```
int main (int argc, char *argv[], char *envp[]);
```

The identifier *argc* is the argument count; *argv* is an array of argument strings. The first member of the array (*argv*[0]) contains the name of the image. The arguments are placed in subsequent elements of the array. The last element of the array is always the NULL pointer.

An `exec` function calls a child process in the same way that the run-time system calls any other *Compaq C* program. The `exec` functions pass the name of the image to be activated in the child; this value is placed in *argv*[0]. However, the functions differ in the way they pass arguments and environment information to the child:

- Arguments can be passed in separate character strings (`exec1`, `execle`, and `exec1p`) or in an array of character strings (`execv`, `execve`, and `execvp`).
- The environment can be explicitly passed in an array (`execle` and `execve`) or taken from the parent's environment (`exec1`, `execv`, `exec1p`, and `execvp`).

If `vfork` was called before invoking an `exec` function, then when the `exec` function completes, control is returned to the parent process at the point of the `vfork` call. If `vfork` was not called, the `exec` function waits until the child has completed execution and then exits the parent process. See `vfork` in this section and Chapter 5 for more information.

**execl**

**Return Value**

–1

Indicates failure.



---

## execle

Passes the name of an image to be activated in a child process. This function is nonreentrant.

### Format

```
#include <unistd.h>

int execle (char *file_spec, char *arg0, . . . , (char *)0, char *envp[]); (ISO POSIX-1)

int execle (char *file_spec, . . .); (Compatibility)
```

### Arguments

#### **file\_spec**

The full file specification of a new image to be activated in the child process.

#### **arg0, ...**

A sequence of pointers to null-terminated character strings.

If the POSIX-1 format is used, at least one argument must be present and must point to a string that is the same as the new process file name (or its last component). (This pointer can also be the NULL pointer, but then `execle` would accomplish nothing.) The last pointer must be the NULL pointer. This is also the convention if the compatibility format is used.

#### **envp**

An array of strings that specifies the program's environment. Each string in *envp* has the following form:

```
name = value
```

The name can be one of the following names and the value is a null-terminated string to be associated with the name:

- HOME—Your login directory
- TERM—The type of terminal being used
- PATH—The default device and directory
- USER—The name of the user who initiated the process

The last element in *envp* must be the NULL pointer.

When the operating system executes the program, it places a copy of the current environment vector (*envp*) in the external variable `environ`.

### Description

See `execl` in this section for a description of how the `exec` functions operate.

### Return Value

–1 Indicates failure.

## execvp

---

## execvp

Passes the name of an image to be activated in a child process. This function is nonreentrant.

### Format

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

```
int execvp (const char *file_name, const char *arg0, . . . , (char *)0); (ISO POSIX-1)
```

```
int execvp (char *file_name, . . .); (Compatibility)
```

### Arguments

#### **file\_name**

The file name of a new image to be activated in the child process. The device and directory specification for the file is obtained by searching the environment name VAXC\$PATH.

#### **argn**

A sequence of pointers to null-terminated character strings. By convention, at least one argument must be present and must point to a string that is the same as the new process file name (or its last component).

. . .

A sequence of pointers to strings. At least one pointer must exist to terminate the list. This pointer must be the NULL pointer.

### Description

See `execl` in this section for a description of how the `exec` functions operate.

### Return Value

|    |                    |
|----|--------------------|
| -1 | Indicates failure. |
|----|--------------------|

---

## execv

Passes the name of an image to be activated in a child process. This function is nonreentrant.

### Format

```
#include <unistd.h>

int execv (char *file_spec, char *argv[]);
```

### Arguments

**file\_spec**

The full file specification of a new image to be activated in the child process.

**argv**

An array of pointers to null-terminated character strings. These strings constitute the argument list available to the new process. By convention, *argv*[0] must point to a string that is the same as the new process file name (or its last component). *argv* is terminated by a NULL pointer.

### Description

See `execl` in this section for a description of how the exec functions operate.

### Return Value

|    |                    |
|----|--------------------|
| –1 | Indicates failure. |
|----|--------------------|

**execve**

Passes the name of an image to be activated in a child process. This function is nonreentrant.

## Format

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

```
int execlve (const char *file_spec, char *argv[], char *envp[]);
```

## Arguments

## file\_spec

The full file specification of a new image to be activated in the child process.

**argv**

An array of pointers to null-terminated character strings. These strings constitute the argument list available to the new process. By convention, *argv*[0] must point to a string that is the same as the new process file name (or its last component). *argv* is terminated by a NULL pointer.

**envp**

An array of strings that specifies the program's environment. Each string in *envp* has the following form:

name = value

The name can be one of the following names and the value is a null-terminated string to be associated with the name:

- HOME—Your login directory
- TERM—The type of terminal being used
- PATH—The default device and directory
- USER—The name of the user who initiated the process

The last element in *envp* must be the NULL pointer.

When the operating system executes the program, it places a copy of the current environment vector (*envp*) in the external variable `environ`.

### Description

See `exec1` in this section for a description of how the `exec` functions operate.

## Return Value

–1 Indicates failure.

---

## execvp

Passes the name of an image to be activated in a child process. This function is nonreentrant.

### Format

```
#include <unistd.h>

int execvp (const char *file_name, char *argv[]);
```

### Arguments

**file\_name**

The file name of a new image to be activated in the child process. The device and directory specification for the file is obtained by searching the environment name VAXC\$PATH.

**argv**

An array of pointers to null-terminated character strings. These strings constitute the argument list available to the new process. By convention, *argv*[0] must point to a string that is the same as the new process file name (or its last component). *argv* is terminated by a NULL pointer.

### Description

See `execl` in this section for a description of how the `exec` functions operate.

### Return Value

|    |                    |
|----|--------------------|
| –1 | Indicates failure. |
|----|--------------------|

## exit, \_exit

---

## exit, \_exit

Terminate execution of the program from which they are called. These functions are nonreentrant.

### Format

```
#include <stdlib.h>

void exit (int status);

#include <unistd.h>

void _exit (int status);
```

### Argument

#### status

A status value of EXIT\_SUCCESS (0), EXIT\_FAILURE (1), or a number from 2 to 255:

- A status value of 0 or EXIT\_SUCCESS is translated to the OpenVMS SS\$\_NORMAL status code to return the OpenVMS success value.
- A status value of 1 or EXIT\_FAILURE is translated to an error-level exit status. The status value is passed to the parent process.
- Any other status value is left the same.

To use these status values as described, include <unistd.h> and compile with the \_POSIX\_EXIT feature-test macro set (either with /DEFINE=\_POSIX\_EXIT or with #define \_POSIX\_EXIT at the top of your file, before any file inclusions). This behavior is available only on OpenVMS Version 7.0 and higher systems.

### Description

If the process was invoked by the DIGITAL Command Language (DCL), the status is interpreted by DCL and a message is displayed.

If the process was a child process created using vfork or an exec function, then the child process exits and control returns to the parent. The two functions are identical; the \_exit function is retained for reasons of compatibility with VAX C.

The exit and \_exit functions make use of the \$EXIT system service. If your process is being invoked by the RUN command using any of the hibernation and scheduled wakeup qualifiers, the process might not correctly return to hibernation state when an exit or \_exit call is made.

---

#### Note

EXIT\_SUCCESS and EXIT\_FAILURE are portable across any ANSI C compiler to indicate success or failure. On OpenVMS systems, they are mapped to OpenVMS condition codes with the severity set to success or failure, respectively. Values in the range of 2 to 255 can be used by a child process to communicate a small amount of data to the parent. The parent retrieves this data using the wait, wait3, wait4, or waitpid functions.

---

---

## exp

Returns the base e raised to the power of the argument.

### Format

```
#include <math.h>

double exp (double x);
float expf (float x); (Alpha only)
long double expl (long double x); (Alpha only)
double expm1 (double x); (Alpha only)
float expm1f (float x); (Alpha only)
long double expm1l (long double x); (Alpha only)
```

### Argument

**x**  
A real value.

### Description

The exp functions compute the value of the exponential function, defined as  $e^{**}x$ , where e is the constant used as a base for natural logarithms.

The expm1 functions compute  $\exp(x) - 1$  accurately, even for tiny  $x$ .

If an overflow occurs, the exp functions return the largest possible floating-point value and set errno to ERANGE. The constant HUGE\_VAL is defined in the <math.h> header file to be the largest possible floating-point value.

### Return Values

|          |                                             |
|----------|---------------------------------------------|
| x        | The exponential value of the argument.      |
| HUGE_VAL | Overflow occurred; errno is set to ERANGE.  |
| 0        | Underflow occurred; errno is set to ERANGE. |
| NaN      | $x$ is NaN; errno is set to EDOM.           |

## **fabs**

---

## **fabs**

Returns the absolute value of its argument.

### **Format**

```
#include <math.h>

double fabs (double x);
float fabsf (float x); (Alpha only)
long double fabsl (long double x); (Alpha only)
```

### **Argument**

**x**  
A real value.

### **Return Value**

|          |                                     |
|----------|-------------------------------------|
| <b>x</b> | The absolute value of the argument. |
|----------|-------------------------------------|



---

## fchown

Changes the owner and group of a file.

### Format

```
#include <unistd.h>

int fchown (int fildes, uid_t owner, gid_t group);
```

### Arguments

**fildes**

An open file descriptor.

**owner**

A user ID corresponding to the new owner of the file.

**group**

A group ID corresponding to the group of the file.

### Description

The `fchown` function has the same effect as `chown` except that the file whose owner and group are to be changed is specified by the file descriptor *fildes*.

### Return Values

0

Indicates success.

−1

Indicates failure. The function sets *errno* to one of the following values:

The `fchown` function *will* fail if:

- **EBADF** – The *fildes* argument is not an open file descriptor.
- **EPERM** – The effective user ID does not match the owner of the file, or the process does not have appropriate privilege.
- **EROFS** – The file referred to by *fildes* resides on a read-only file system.

The `fchown` function *may* fail if:

- **EINVAL** – The owner or group ID is not a value supported by the implementation.
- **EIO** – A physical I/O error has occurred.
- **EINTR** – The `fchown` function was interrupted by a signal that was caught.

## fclose

---

## fclose

Closes a file by flushing any buffers associated with the file control block and freeing the file control block and buffers previously associated with the file pointer.

### Format

```
#include <stdio.h>

int fclose (FILE *file_ptr);
```

### Argument

**file\_ptr**

A pointer to the file to be closed.

### Description

When a program terminates normally, the `fclose` function is automatically called for all open files.

The `fclose` function tries to write buffered data by using an implicit call to `fflush`.

If the write fails (because the disk is full or the user's quota is exceeded, for example), `fclose` continues executing. It closes the VMS channel, deallocates any buffers, and releases the memory associated with the file descriptor (or `FILE` pointer). Any buffered data is lost, and the file descriptor (or `FILE` pointer) no longer refers to the file.

If your program needs to recover from errors when flushing buffered data, it should make an explicit call to `fsync` (or `fflush`) before calling `fclose`.

### Return Values

0

Indicates success.

EOF

Indicates that the file control block is not associated with an open file.

---

**fcntl**

Performs controlling operations on an open file.

**Format**

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

int fcntl (int file_desc, int request [, int file_desc2]);
```

**Arguments****file\_desc**

An open file descriptor obtained from a successful open, fcntl, or pipe function.

**request**

The operation to be performed.

**file\_desc2**

A variable that depends on the value of the *request* argument.

**Description**

The fcntl function performs controlling operations on the open file specified by the *file\_desc* argument.

The values for the *request* argument are defined in the header file <fcntl.h>, and include the following:

**F\_DUPFD**

Returns a new file descriptor that is the lowest numbered available (that is, not already open) file descriptor greater than or equal to the third argument (*file\_desc2*) taken as an integer of type int.

The new file descriptor refers to the same file as the original file descriptor (*file\_desc*). The FD\_CLOEXEC flag associated with the new file descriptor is cleared to keep the file open across calls to one of the exec functions.

The following two calls are equivalent:

```
fid = dup(file_desc);

fid = fcntl(file_desc, F_DUPFD, 0);
```

The following call

```
fid = dup2(file_desc, file_desc2);
```

is similar (but not equivalent) to:

```
close(file_desc2);
fid = fcntl(file_desc, F_DUPFD, file_desc2);
```

## fcntl

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| F_GETFD | Gets the value of the close-on-exec flag associated with the file descriptor <i>file_desc</i> . File descriptor flags are associated with a single file descriptor and do not affect other file descriptors that refer to the same file. The <i>file_desc2</i> argument should not be specified.                                                                                                                                                                                                                                                                                                                                                   |
| F_SETFD | <p>Sets the close-on-exec flag associated with <i>file_desc</i> to the value of the third argument, taken as type <code>int</code>:</p> <p>If the third argument is 0, the file remains open across the <code>exec</code> functions, which means that a child process spawned by the <code>exec</code> function inherits this file descriptor from the parent.</p> <p>If the third argument is <code>FD_CLOEXEC</code>, the file is closed on successful execution of the next <code>exec</code> function, which means that the child process spawned by the <code>exec</code> function will not inherit this file descriptor from the parent.</p> |

## Return Values

|   |                                                                                                                                                                                                                                                                                                                                                                                   |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n | <p>Upon successful completion, the value returned depends on the value of the <i>request</i> argument as follows:</p> <ul style="list-style-type: none"><li>• <code>F_DUPFD</code> – Returns a new file descriptor.</li><li>• <code>F_GETFD</code> – Returns <code>FD_CLOEXEC</code> or 0.</li><li>• <code>F_SETFD</code> – Returns a value other than <code>-1</code>.</li></ul> |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

-1

Indicates that an error occurred. The function sets *errno* to the following value:

- **EBADF** – The *file\_desc* argument is not a valid open file descriptor and the *file\_desc2* argument is negative or greater than or equal to the per-process limit.
- **EFAULT** – The *file\_desc2* argument is an invalid address.
- **EINVAL** – The *request* argument is `F_DUPFD` and the *file\_desc2* argument is negative or greater than or equal to `OPEN_MAX`.  
Either the `OPEN_MAX` value or the per-process soft descriptor limit is checked.  
An illegal value was provided for the *request* argument.
- **EMFILE** – The *request* argument is `F_DUPFD` and too many or `OPEN_MAX` file descriptors are currently open in the calling process, or no file descriptors greater than or equal to argument are available.  
Either the `OPEN_MAX` value or the per-process soft descriptor limit is checked.
- **ENOMEM** – The system was unable to allocate memory for the requested file descriptor.

---

## fcvt

Converts its argument to a null-terminated string of ASCII digits and returns the address of the string. The string is stored in a thread-specific location created by the *Compaq C* RTL.

### Format

```
#include <stdlib.h>

char *fcvt (double value, int ndigits, int *decpt, int *sign);
```

### Arguments

**value**

An object of type `double` that is converted to a null-terminated string of ASCII digits.

**ndigits**

The number of ASCII digits after the decimal point to be used in the converted string.

**decpt**

The position of the decimal point relative to the first character in the returned string. The returned string does not contain the actual decimal point. A negative `int` value means that the decimal point is *decpt* number of spaces to the left of the returned digits (the spaces are filled with zeros). A 0 value means that the decimal point is immediately to the left of the first digit in the returned string.

**sign**

An integer value that indicates whether the *value* argument is positive or negative. If *value* is negative, the `fcvt` function places a nonzero value at the address specified by *sign*. Otherwise, the functions assign 0 to the address specified by *sign*.

### Description

The `fcvt` function converts *value* to a null-terminated string and returns a pointer to it. The resulting low-order digit is rounded to the correct digit for outputting *ndigits* digits in C F-format. The *decpt* argument is assigned the position of the decimal point relative to the first character in the string.

In C F-format, *ndigits* is the number of digits desired after the decimal point. Very large numbers produce a very long string of digits before the decimal point, and *ndigit* of digits after the decimal point. For large numbers, it is preferable to use the `gcv` or `ecvt` function so that E-format is used.

Repeated calls to the `fcvt` function overwrite any existing string.

The `ecvt`, `fcvt`, and `gcv` functions represent the following special values specified in the IEEE Standard for floating-point arithmetic:

---

| Value     | Representation |
|-----------|----------------|
| Quiet NaN | NaNQ           |

---

| Value          | Representation |
|----------------|----------------|
| Signalling NaN | NaNs           |
| +Infinity      | Infinity       |
| − Infinity     | − Infinity     |

The sign associated with each of these values is stored into the *sign* argument. In IEEE floating-point representation, a value of 0 (zero) can be positive or negative, as set by the *sign* argument.

See also `gcvt` and `ecvt` in this section.

## Return Value

|   |                                    |
|---|------------------------------------|
| x | A pointer to the converted string. |
|---|------------------------------------|

## fdopen

---

## fdopen

Associates a file pointer with a file descriptor returned by an `open`, `creat`, `dup`, `dup2`, or `pipe` function.

### Format

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

```
FILE *fdopen (int file_desc, char *a_mode);
```

### Arguments

#### **file\_desc**

The file descriptor returned by `open`, `creat`, `dup`, `dup2`, or `pipe`.

#### **a\_mode**

The access mode indicator. See the `fopen` function in this section for a description. Note that the access mode specified must agree with the mode used to open the file originally. This includes binary/text access mode ("b" mode on `fdopen` and the "ctx=bin" option on `creat` or `open`).

### Description

The `fdopen` function allows you to access a file, originally opened by one of the UNIX I/O functions, with Standard I/O functions. Ordinarily, a file can be accessed by either a file descriptor or by a file pointer, but not both, depending on the way you open it. For more information, see Chapters 1 and 2.

### Return Values

pointer

Indicates that the operation has succeeded.

NULL

Indicates that an error has occurred.



---

## feof

Tests a file to see if the end-of-file has been reached.

### Format

```
#include <stdio.h>
int feof (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A file pointer.

### Return Values

|                 |                                                      |
|-----------------|------------------------------------------------------|
| nonzero integer | Indicates that the end-of-file has been reached.     |
| 0               | Indicates that the end-of-file has not been reached. |

## feof

---

## feof

Returns a nonzero integer if an error occurred while reading or writing a file.

### Format

```
#include <stdio.h>

int feof (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A file pointer.

### Description

A call to feof continues to return a nonzero integer until the file is closed or until clearerr is called.

### Return Values

|                 |                                       |
|-----------------|---------------------------------------|
| 0               | Indicates success.                    |
| nonzero integer | Indicates that an error has occurred. |

---

## fflush

Writes out any buffered information for the specified file.

### Format

```
#include <stdio.h>

int fflush (FILE *file_ptr);
```

### Argument

**file\_ptr**

A file pointer. If this argument is a NULL pointer, all buffers associated with all currently open files are flushed.

### Description

The output files are normally buffered only if they are not directed to a terminal, except for stderr, which is not buffered by default.

The fflush function flushes the *Compaq C* RTL buffers. However, RMS has its own buffers. The fflush function does not guarantee that the file will be written to disk. (See the description of fsync for a way to flush buffers to disk.)

If the file pointed to by *file\_ptr* was opened in record mode and if there is unwritten data in the buffer, then fflush always generates a record.

### Return Values

|     |                                                                                                                                       |
|-----|---------------------------------------------------------------------------------------------------------------------------------------|
| 0   | Indicates that the operation is successful.                                                                                           |
| EOF | Indicates that the buffered data cannot be written to the file, or that the file control block is not associated with an output file. |

## ffs

---

## ffs

Finds the index of the first bit set in a string.

### Format

```
#include <strings.h>
int ffs (int integer);
```

### Arguments

**integer**

The integer to be examined for the first bit set.

### Description

The `ffs` function finds the first bit set (beginning with the least significant bit) and returns the index of that bit. Bits are numbered starting at 1 (the least significant bit).

### Return Values

|   |                                 |
|---|---------------------------------|
| x | The index of the first bit set. |
| 0 | If <i>index</i> is 0.           |

---

## fgetc

Returns the next character from a specified file.

### Format

```
#include <stdio.h>
int fgetc (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A pointer to the file to be accessed.

### Description

See the `getc` macro in this section.

### Return Values

|     |                                        |
|-----|----------------------------------------|
| x   | The returned character.                |
| EOF | Indicates the end-of-file or an error. |

## fgetname

---

## fgetname

Returns the file specification associated with a file pointer.

### Format

```
#include <stdio.h>

char *fgetname (FILE *file_ptr, char *buffer, ...);
```

### Function Variants

The `fgetname` function has variants named `_fgetname32` and `_fgetname64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**file\_ptr**

A file pointer.

**buffer**

A pointer to a character string that is large enough to hold the file specification.

...

An optional additional argument that can be either 1 or 0. If you specify 1, the `fgetname` function returns the file specification in OpenVMS format. If you specify 0, `fgetname` returns the file specification in UNIX style format. If you do not specify this argument, `fgetname` returns the file name according to your current command language interpreter. For more information about UNIX style file specifications, see Section 1.4.3.

### Description

The `fgetname` function places the file specification at the address given in the buffer. The buffer should be an array large enough to contain a fully qualified file specification (the maximum length is 256 characters).

### Return Values

|   |                            |
|---|----------------------------|
| n | The address of the buffer. |
| 0 | Indicates an error.        |

### Restriction

The `fgetname` function is specific to the *Compaq C* RTL and is not portable.

## fgetpos

Stores the current file position for a given file.

### Format

```
#include <stdio.h>

int fgetpos (FILE *stream, fpos_t *pos);
```

### Arguments

#### **stream**

A file pointer.

#### **pos**

A pointer to an implementation-defined structure. The fgetpos function fills this structure with information that can be used on subsequent calls to fsetpos.

### Description

The fgetpos function stores the current value of the file position indicator for the stream pointed to by *stream* into the object pointed to by *pos*.

### Return Values

|    |                                  |
|----|----------------------------------|
| 0  | Indicates successful completion. |
| -1 | Indicates that there are errors. |

### Example

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

main()
{
 FILE *fp;
 int stat,
 i;
 int character;
 char ch,
 c_ptr[130],
 d_ptr[130];
 fpos_t posit;

 /* Open a file for writing. */
 if ((fp = fopen("file.dat", "w+")) == NULL) {
 perror("open");
 exit(1);
 }

 /* Get the beginning position in the file. */
 if (fgetpos(fp, &posit) != 0)
 perror("fgetpos");

 /* Write some data to the file. */
 if (fprintf(fp, "this is a test\n") == 0) {
 perror("fprintf");
 exit(1);
 }
}
```

## fgetpos

```
/* Set the file position back to the beginning. */
if (fsetpos(fp, &posit) != 0)
 perror("fsetpos");
fgets(c_ptr, 130, fp);
puts(c_ptr); /* Should be "this is a test." */
/* Close the file. */
if (fclose(fp) != 0) {
 perror("close");
 exit(1);
}
}
```



---

## fgets

Reads a line from the specified file, up to one less than the specified maximum number of characters or up to and including the new-line character, whichever comes first. The function stores the string in *str*.

### Format

```
#include <stdio.h>

char *fgets (char *str, int maxchar, FILE *file_ptr);
```

### Function Variants

The `fgets` function has variants named `_fgets32` and `_fgets64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **str**

A pointer to a character string that is large enough to hold the information fetched from the file.

#### **maxchar**

The maximum number of characters to fetch.

#### **file\_ptr**

A file pointer.

### Description

The `fgets` function terminates the line with a null character (`\0`). Unlike `gets`, `fgets` places the new-line character that terminates the input line into the user buffer if more than *maxchar* characters have not already been fetched.

When the file pointed to by *file\_ptr* is opened in record mode, `fgets` treats the end of a record the same as a new-line character, so it reads up to and including a new-line character or to the end of the record.

### Return Values

|                   |                                                                                                         |
|-------------------|---------------------------------------------------------------------------------------------------------|
| <code>x</code>    | Pointer to <i>str</i> .                                                                                 |
| <code>NULL</code> | Indicates the end-of-file or an error. The contents of <i>str</i> are undefined if a read error occurs. |

### Example

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

main()
{
 FILE *fp;
 char c_ptr[130];
 /* Create a dummy data file */
```

## fgets

```
if ((fp = fopen("file.dat", "w+")) == NULL) {
 perror("open");
 exit(1);
}

fprintf(fp, "this is a test\n") ;
fclose(fp) ;

/* Open a file with some data -"this is a test" */
if ((fp = fopen("file.dat", "r+")) == NULL) {
 perror("open error") ;
 exit(1);
}

fgets(c_ptr, 130, fp);
puts(c_ptr); /* Display what fgets got. */
fclose(fp);

delete("file.dat") ;
}
```

---

## fgetwc

Reads the next character from a specified file, and converts it to a wide-character code.

### Format

```
#include <wchar.h>

wint_t fgetwc (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A pointer to the file to be accessed.

### Description

Upon successful completion, the `fgetwc` function returns the wide-character code read from the file pointed to by *file\_ptr* and converted to type `wint_t`. If the file is at end-of-file, the end-of-file indicator is set, and `WEOF` is returned. If an I/O read error occurred, then the error indicator is set, and `WEOF` is returned.

Applications can use `ferror` or `feof` to distinguish between an error condition and an end-of-file condition.

### Return Values

|                   |                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>    | The wide-character code of the character read.                                                                                                                                                                                                                                                                                                                                            |
| <code>WEOF</code> | Indicates the end-of-file or an error. If a read error occurs, the function sets <code>errno</code> to one of the following: <ul style="list-style-type: none"><li>• <code>EALREADY</code> – An operation is already in progress on the same file.</li><li>• <code>EBADF</code> – The file descriptor is not valid.</li><li>• <code>EILSEQ</code> – Invalid character detected.</li></ul> |

## fgetws

---

## fgetws

Reads a line of wide characters from a specified file.

### Format

```
#include <wchar.h>

wchar_t *fgetws (wchar_t *wstr, int maxchar, FILE *file_ptr);
```

### Function Variants

The `fgetws` function has variants named `_fgetws32` and `_fgetws64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **wstr**

A pointer to a wide-character string large enough to hold the information fetched from the file.

#### **maxchar**

The maximum number of wide characters to fetch.

#### **file\_ptr**

A file pointer.

### Description

The `fgetws` function reads wide characters from the specified file and stores them in the array pointed to by `wstr`. The function reads up to `maxchar-1` characters or until the new-line character is read, converted, and transferred to `wstr`, or until an end-of-file condition is encountered. The function terminates the line with a null wide character. `fgetws` places the new-line that terminates the input line into the user buffer, unless `maxchar` characters have already been fetched.

### Return Values

`x`

Pointer to `wstr`.

`NULL`

Indicates the end-of-file or an error occurred. The contents of `wstr` are undefined if a read error occurs. If a read error occurs, the function sets `errno`. For a list of possible `errno` values, see `fgetc` in this section.

## Example

```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>
#include <wchar.h>

main()
{
 wchar_t wstr[80],
 *ret;
 FILE *fp;

 /* Create a dummy data file */
 if ((fp = fopen("file.dat", "w+")) == NULL) {
 perror("open");
 exit(1);
 }

 fprintf(fp, "this is a test\n") ;
 fclose(fp) ;

 /* Open a test file containing : "this is a test" */
 if ((fp = fopen("file.dat", "r")) == (FILE *) NULL) {
 perror("File open error");
 exit(EXIT_FAILURE);
 }

 ret = fgetws(wstr, 80, fp);
 if (ret == (wchar_t *) NULL) {
 perror("fgetws failure");
 exit(EXIT_FAILURE);
 }

 fputws(wstr, stdout);
 fclose(fp);
 delete("file.dat");
}
```

## fileno

---

## fileno

Returns the file descriptor associated with the specified file pointer.

### Format

```
#include <stdio.h>

int fileno (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A file pointer.

### Description

If you are using DEC C Version 5.2 or lower, undefine the `fileno` macro:

```
#if defined(fileno)
#undef fileno
#endif
```

### Return Values

|    |                          |
|----|--------------------------|
| x  | Integer file descriptor. |
| −1 | Indicates an error.      |

---

**finite** *(Alpha only)*

Returns the integer value 1 (TRUE) when its argument is a finite number, or 0 (FALSE) if not.

**Format**

```
#include <math.h>

int finite (double x);

int finitf (float x);

int double finitel (long double x);
```

**Argument**

**x**  
A real value.

**Description**

The `finite` functions return 1 when  $-\text{Infinity} < x < +\text{Infinity}$ . They return 0 when  $|x| = \text{Infinity}$ , or  $x$  is a NaN.

## floor

---

## floor

Returns the largest integer less than or equal to the argument.

### Format

```
#include <math.h>

double floor (double x);
float floorf (float x); (Alpha only)
long double floorl (long double x); (Alpha only)
```

### Argument

**x**  
A real value.

### Return Value

|   |                                                         |
|---|---------------------------------------------------------|
| n | The largest integer less than or equal to the argument. |
|---|---------------------------------------------------------|



---

## fmod

Computes the floating-point remainder.

### Format

```
#include <math.h>

double fmod (double x, double y);
float fmodf (float x, float y); (Alpha only)
long double fmodl (long double x, long double y); (Alpha only)
```

### Arguments

**x**  
A real value.

**y**  
A real value.

### Description

The fmod functions return the floating-point remainder of the first argument divided by the second. If the second argument is 0, the function returns 0.

### Return Values

|   |                                                                                                                                                                                |
|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x | The value $f$ , which has the same sign as the argument $x$ , such that $x == i * y + f$ for some integer $i$ , where the magnitude of $f$ is less than the magnitude of $y$ . |
| 0 | Indicates that $y$ is 0.                                                                                                                                                       |

## fopen

---

## fopen

Opens a file by returning the address of a FILE structure.

### Format

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

```
FILE *fopen (const char *file_spec, const char *a_mode); (ANSI C)
```

```
FILE *fopen (const char *file_spec, const char *a_mode, ...); (COMPAQ C Extension)
```

### Arguments

#### file\_spec

A character string containing a valid file specification.

#### a\_mode

The access mode indicator. Use one of the following character strings: "r", "w", "a", "r+", "w+", "rb", "r+b", "rb+", "wb", "w+b", "wb+", "ab", "a+b", "ab+", or "a+".

These access modes have the following effects:

- "r" opens an existing file for reading.
- "w" creates a new file, if necessary, and opens the file for writing. If the file exists, it creates a new file with the same name and a higher version number.
- "a" opens the file for append access. An existing file is positioned at the end-of-file, and data is written there. If the file does not exist, the *Compaq C* RTL creates it.

The update access modes allow a file to be opened for both reading and writing. When used with existing files, "r+" and "a+" differ only in the initial positioning within the file. The modes are as follows:

- "r+" opens an existing file for read update access. It is opened for reading, positioned first at the beginning-of-file, but writing is also allowed.
- "w+" opens a new file for write update access.
- "a+" opens a file for append update access. The file is first positioned at the end-of-file (writing). If the file does not exist, the *Compaq C* RTL creates it.
- "b" means binary access mode. In this case, no conversion of carriage-control information is attempted.

...

Optional file attribute arguments. The file attribute arguments are the same as those used in the *creat* function. For more information, see the *creat* function.

### Description

If a version of the file exists, a new file created with *fopen* inherits certain attributes from the existing file unless those attributes are specified in the *fopen* call. The following attributes are inherited:

- Record format
- Maximum record size
- Carriage control

### File protection

If you specify a directory in the file name and it is a search list that contains an error, *Compaq C* for *OpenVMS* Systems interprets it as a file open error.

The file control block can be freed with the `fclose` function, or by default on normal program termination.

## Return Values

`x`

File pointer.

`NULL`

Indicates an error. The constant `NULL` is defined in the `<stdio.h>` header file to be the `NULL` pointer value. The function returns `NULL` to signal the following errors:

- File protection violations
- Attempts to open a nonexistent file for read access
- Failure to open the specified file

**fp\_class** (*Alpha only*)**fp\_class** (*Alpha only*)

Determine the class of IEEE floating-point values.

## Format

```
#include <math.h>

int fp_class (double x);

int fp_classf (float x);

int fp_classl (long double x);
```

## Argument

**x**  
An IEEE floating-point number.

## Description

These functions determine the class of the specified IEEE floating-point number, returning a constant from the `<fp_class.h>` header file. They never cause an exception, even for signaling NaNs (Not-a-Number). These functions implement the recommended `class(x)` function in the appendix of the IEEE 754-1985 standard for binary floating-point arithmetic. The constants in `<fp_class.h>` refer to the following classes of values:

|               |                              |
|---------------|------------------------------|
| FP_SNAN       | Signaling NaN (Not-a-Number) |
| FP_QNAN       | Quiet NaN                    |
| FP_POS_INF    | +Infinity                    |
| FP_NEG_INF    | −Infinity                    |
| FP_POS_NORM   | positive normalized          |
| FP_NEG_NORM   | negative normalized          |
| FP_POS_DENORM | positive denormalized        |
| FP_NEG_DENORM | negative denormalized        |
| FP_POS_ZERO   | +0.0 (positive zero)         |
| FP_NEG_ZERO   | −0.0 (negative zero)         |

## Return Value

|          |                                                                  |
|----------|------------------------------------------------------------------|
| <b>x</b> | A constant from the <code>&lt;fp_class.h&gt;</code> header file. |
|----------|------------------------------------------------------------------|

---

## fpathconf

Retrieves file implementation characteristics.

### Format

```
#include <unistd.h>

long int fpathconf (int filesdes, int name);
```

### Arguments

#### **filesdes**

An open file descriptor.

#### **name**

The configuration attribute to query. If this attribute is not applicable to the file specified by the *filesdes* argument, fpathconf returns an error.

### Description

The fpathconf function allows an application to retrieve the characteristics of operations supported by the file system underlying the file named by the *filesdes* argument. Read, write, or execute permission of the named file is not required, but you must be able to search all directories in the path leading to the file.

Symbolic values for the *name* argument are defined in the <unistd.h> header file as follows:

|                            |                                                                                                                                                                                                                                                                                                    |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_PC_LINK_MAX</code>  | The maximum number of links to the file. If the <i>filesdes</i> argument refers to a directory, the value returned applies to the directory itself.                                                                                                                                                |
| <code>_PC_MAX_CANON</code> | The maximum number of bytes in a canonical input line. This is applicable only to terminal devices.                                                                                                                                                                                                |
| <code>_PC_MAX_INPUT</code> | The number of types allowed in an input queue. This is applicable only to terminal devices.                                                                                                                                                                                                        |
| <code>_PC_NAME_MAX</code>  | Maximum number of bytes in a file name (not including a terminating null). The byte range value is between 13 and 255. This is applicable only to a directory file. The value returned applies to filenames within the directory.                                                                  |
| <code>_PC_PATH_MAX</code>  | Maximum number of bytes in a path name (not including a terminating null). The value is never larger than 65,535. This is applicable only to a directory file. The value returned is the maximum length of a relative pathname when the specified directory is the working directory.              |
| <code>_PC_PIPE_BUF</code>  | Maximum number of bytes guaranteed to be written atomically. This is applicable only to a FIFO. The value returned applies to the referenced object. If the <i>path</i> argument refers to a directory, the value returned applies to any FIFO that exists or can be created within the directory. |

## fpathconf

|                                   |                                                                                                                                                                                                            |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_PC_CHOWN_RESTRICTED</code> | The value returned applies to any files (other than directories) that exist or can be created within the directory. This is applicable only to a directory file.                                           |
| <code>_PC_NO_TRUNC</code>         | Returns 1 if supplying a component name longer than allowed by <code>NAME_MAX</code> causes an error. Returns 0 (zero) if long component names are truncated. This is applicable only to a directory file. |
| <code>_PC_VDISABLE</code>         | This is always 0 (zero); no disabling character is defined. This is applicable only to a terminal device.                                                                                                  |

## Return Values

|                 |                                                                                                                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>  | The resultant value for the configuration attribute specified in the <i>name</i> argument.                                                                                                                                                                                                                                   |
| <code>-1</code> | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"><li>• <code>EINVAL</code> – The <i>name</i> argument specifies an unknown or inapplicable characteristic.</li><li>• <code>EBADF</code> – the <i>files</i> argument is not a valid file descriptor.</li></ul> |

---

## fprintf

Performs formatted output to a specified file.

### Format

```
#include <stdio.h>

int fprintf (FILE *file_ptr, const char *format_spec, ...);
```

### Arguments

**file\_ptr**

A pointer to the file to which the output is directed.

**format\_spec**

A pointer to a character string that contains the format specification. For more information on format specifications and conversion characters, see Chapter 2.

...

Optional expressions whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, the output sources can be omitted. Otherwise, the function calls must have exactly as many output sources as there are conversion specifications, and the conversion specifications must match the types of the output sources.

Conversion specifications are matched to output sources in left-to-right order. Any excess output sources are ignored.

### Description

An example of a conversion specification follows:

```
#include <stdio.h>

main()
{
 int temp = 4, temp2 = 17;

 fprintf(stdout, "The answers are %d, and %d.", temp, temp2);
}
```

Sample output (to the stdout file) from the previous example is as follows:

The answers are 4, and 17.

For a complete description of the format specification and the output source, see Chapter 2.

### Return Values

x

The number of bytes written, excluding the null terminator.

## fprintf

Negative value

Indicates an error. The function sets `errno` to one of the following:

- `EILSEQ` – Invalid character detected.
- `EINVAL` – Insufficient arguments.
- `ENOMEM` – Not enough memory available for conversion.
- `ERANGE` – Floating-point calculations overflow.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This might indicate that conversion to a numeric value failed because of overflow.

The function can also set `errno` to the following as a result of errors returned from the I/O subsystem:

- `EBADF` – The file descriptor is not valid.
- `EIO` – I/O error.
- `ENOSPC` – No free space on the device containing the file.
- `ENXIO` – Device does not exist.
- `EPIPE` – Broken pipe.
- `ESPIPE` – Illegal seek in a file opened for append.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This indicates that an I/O error occurred for which there is no equivalent C error code.



---

## fputc

Writes a character to a specified file.

### Format

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

```
int fputc (int character, FILE *file_ptr);
```

### Arguments

**character**

An object of type `int`.

**file\_ptr**

A file pointer.

### Description

The `fputc` function writes a single character to a file and returns the character. See also `putc` in this section.

### Return Values

|     |                                                       |
|-----|-------------------------------------------------------|
| x   | The character written to the file. Indicates success. |
| EOF | Indicates an output error.                            |

## fputs

---

## fputs

Writes a character string to a file without copying the string's null terminator (`\0`).

### Format

```
#include <stdio.h>

int fputs (const char *str, FILE *file_ptr);
```

### Arguments

**str**  
A pointer to a character string.

**file\_ptr**  
A file pointer.

### Description

See also `puts` in this section. Unlike `puts`, the `fputs` function does not append a new-line character to the output string.

### Return Values

|                   |                     |
|-------------------|---------------------|
| Nonnegative value | Indicates success.  |
| EOF               | Indicates an error. |

---

## fputc

Converts a wide character to its corresponding multibyte value, and writes the result to a specified file.

### Format

```
#include <wchar.h>

wint_t fputc (wint_t wc, FILE *file_ptr);
```

### Arguments

**wc**  
An object of type `wint_t`.

**file\_ptr**  
A file pointer.

### Description

The `fputc` function writes a wide character to a file and returns the character. See also `putc` in this section.

### Return Values

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>    | The character written to the file. Indicates success.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>WEOF</code> | <p>Indicates an output error. The function sets <code>errno</code> to the following:</p> <ul style="list-style-type: none"><li>• <code>EILSEQ</code> – Invalid wide-character code detected.</li></ul> <p>The function can also set <code>errno</code> to the following as a result of errors returned from the I/O subsystem:</p> <ul style="list-style-type: none"><li>• <code>EBADF</code> – The file descriptor is not valid.</li><li>• <code>EIO</code> – I/O error.</li><li>• <code>ENOSPC</code> – No free space on the device containing the file.</li><li>• <code>ENXIO</code> – Device does not exist.</li><li>• <code>EPIPE</code> – Broken pipe.</li><li>• <code>ESPIPE</code> – Illegal seek in a file opened for append.</li><li>• <code>EVMSError</code> – Nontranslatable VMS error. <code>vaxc\$errno</code> contains the VMS error code. This indicates that an I/O error occurred for which there is no equivalent C error code.</li></ul> |

## fputws

---

## fputws

Writes a wide-character string to a file without copying the null-terminating character.

### Format

```
#include <wchar.h>

int fputws (const wchar_t *wstr, FILE *file_ptr);
```

### Arguments

**wstr**  
A pointer to a wide-character string.

**file\_ptr**  
A file pointer.

### Description

The `fputws` function converts the specified wide-character string to a multibyte character string and writes it to the specified file. The function does not append a terminating null byte corresponding to the null wide-character to the output string.

### Return Values

|                   |                                                                                                                               |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------|
| Nonnegative value | Indicates success.                                                                                                            |
| −1                | Indicates an error. The function sets <code>errno</code> . For a list of the values, see <code>fputwc</code> in this section. |

---

## fread

Reads a specified number of items from the file.

### Format

```
#include <stdio.h>

size_t fread (void *ptr, size_t size_of_item, size_t number_items, FILE *file_ptr);
```

### Arguments

**ptr**

A pointer to the location, within memory, where you place the information being read. The type of the object pointed to is determined by the type of the item being read.

**size\_of\_item**

The size of the items being read, in bytes.

**number\_items**

The number of items to be read.

**file\_ptr**

A pointer that indicates the file from which the items are to be read.

### Description

The type `size_t` is defined in the header file `<stdio.h>` as follows:

```
typedef unsigned int size_t
```

The reading begins at the current location in the file. The items read are placed in storage beginning at the location given by the first argument. You must also specify the size of an item, in bytes.

If the file pointed to by *file\_ptr* was opened in record mode, `fread` will read *size\_of\_item* multiplied by *number\_items* bytes from the file. That is, it does not necessarily read *number\_items* records.

### Return Values

|   |                                                           |
|---|-----------------------------------------------------------|
| n | The number of bytes read divided by <i>size_of_item</i> . |
| 0 | Indicates the end-of-file or an error.                    |

## free

---

## free

Makes available for reallocation the area allocated by a previous `calloc`, `malloc`, or `realloc` call.

### Format

```
#include <stdlib.h>

void free (void *ptr);
```

### Argument

**ptr**

The address returned by a previous call to `malloc`, `calloc`, or `realloc`. If *ptr* is a NULL pointer, no action occurs.

### Description

The ANSI C standard defines `free` as not returning a value; therefore, the function prototype for `free` is declared with a return type of `void`. However, since a `free` can fail, and since previous versions of the *Compaq C* RTL have declared `free` to return an `int`, the implementation of `free` does return 0 on success and `-1` on failure.

---

## freopen

Substitutes the file named by a file specification for the open file addressed by a file pointer. The latter file is closed.

### Format

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

```
FILE *freopen (const char *file_spec, const char *a_mode, FILE *file_ptr, ...);
```

### Arguments

**file\_spec**

A pointer to a string that contains a valid OpenVMS or UNIX style file specification. After the function call, the given file pointer is associated with this file.

**a\_mode**

The access mode indicator. See the fopen function in this section for a description.

**file\_ptr**

A file pointer.

...

Optional file attribute arguments. The file attribute arguments are the same as those used in the creat function.

### Description

The freopen function is typically used to associate one of the predefined names stdin, stdout, or stderr with a file. For more information about these predefined names, see Chapter 2.

### Return Values

|          |                                             |
|----------|---------------------------------------------|
| file_ptr | The file pointer, if freopen is successful. |
| NULL     | Indicates an error.                         |

## frexp

---

## frexp

Calculates the fractional and exponent parts of a floating-point value.

### Format

```
#include <math.h>

double frexp (double value, int *eptr);
float frexp (float value, int *eptr); (Alpha only)
long double frexp (long double value, int *eptr); (Alpha only)
```

### Arguments

#### value

A floating-point number of type double, float, or long double.

#### eptr

A pointer to an int where frexp places the exponent.

### Description

The frexp functions break the floating-point number (*value*) into a normalized fraction and an integral power of 2, as follows:

$$\text{value} = \text{fraction} * (2^{\text{exp}})$$

The fractional part is returned as the return value. The exponent is placed in the integer variable pointed to by *eptr*.

### Example

```
#include <math.h>

main ()
{
 double val = 16.0, fraction;
 int exp;

 fraction = frexp(val, &exp);
 printf("fraction = %f\n", fraction);
 printf("exp = %d\n", exp);
}
```

In this example, frexp converts the value 16 to  $.5 * 2^5$ . The example produces the following output:

```
fraction = 0.500000
exp = 5
```

| *value* | = Infinity or NaN is an invalid argument.



**Return Values**

|              |                                                                                                                                             |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>x</i>     | The fractional part of <i>value</i> .                                                                                                       |
| 0            | Both parts of the result are 0.                                                                                                             |
| NaN          | If <i>value</i> is NaN, NaN is returned, <code>errno</code> is set to EDOM, and the value of <i>*eptr</i> is unspecified.                   |
| <i>value</i> | If $ value  = \text{Infinity}$ , <i>value</i> is returned, <code>errno</code> is set to EDOM, and the value of <i>*eptr</i> is unspecified. |

---

**fscanf**

Performs formatted input from a specified file, interpreting it according to the format specification.

**Format**

```
#include <stdio.h>

int fscanf (FILE *file_ptr, const char *format_spec, ...);
```

**Arguments****file\_ptr**

A pointer to the file that provides input text.

**format\_spec**

A pointer to a character string that contains the format specification. For more information on conversion characters, see Chapter 2.

...

Optional expressions whose results correspond to conversion specifications given in the format specification.

If no conversion specifications are given, you can omit the input pointers. Otherwise, the function calls must have exactly as many input pointers as there are conversion specifications, and the conversion specifications must match the types of the input pointers.

Conversion specifications are matched to input sources in left-to-right order. Excess input pointers, if any, are ignored.

**Description**

An example of a conversion specification follows:

```
#include <stdio.h>

main ()
{
 int temp, temp2;

 fscanf(stdin, "%d %d", &temp, &temp2);
 printf("The answers are %d, and %d.", temp, temp2);
}
```

Consider a file, designated by `stdin`, with the following contents:

```
4 17
```

The example conversion specification produces the following result:

The answers are 4, and 17.

For a complete description of the format specification and the input pointers, see Chapter 2.

## Return Values

x

The number of successfully matched and assigned input items.

EOF

Indicates that the end-of-file was encountered or a read error occurred. If a read error occurs, the function sets `errno` to one of the following:

- EILSEQ – Invalid character detected.
- EVMSERR – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This can indicate that conversion to a numeric value failed due to overflow.

The function can also set `errno` to the following as a result of errors returned from the I/O subsystem:

- EBADF – The file descriptor is not valid.
- EIO – I/O error.
- ENXIO – Device does not exist.
- EPIPE – Broken pipe.
- EVMSERR – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This indicates that an I/O error occurred for which there is no equivalent C error code.

---

## fseek

Positions the file to the specified byte offset in the file.

### Format

```
#include <stdio.h>

int fseek (FILE *file_ptr, long int offset, int direction);
```

### Arguments

**file\_ptr**

A file pointer.

**offset**

The offset, specified in bytes.

**direction**

An integer indicating the position to which the *offset* is added to calculate the new position. The new position is the beginning of the file if *direction* is `SEEK_SET`, the current value of the file position indicator if *direction* is `SEEK_CUR`, or end-of-file if *direction* is `SEEK_END`.

### Description

The `fseek` function can position fixed-length record-access file with no carriage control or a stream-access file on any byte offset, but can position all other files only on record boundaries.

The available Standard I/O functions position a variable-length or VFC record file at its first byte, at the end-of-file, or on a record boundary. Therefore, the arguments given to `fseek` must specify any of the following:

- The beginning or end of the file
- A 0 offset from the current position (an arbitrary record boundary)
- The position returned by a previous, valid `ftell` call

See the `fgetpos` and `fsetpos` functions for a portable way to seek to arbitrary locations with these types of record files.

---

#### CAUTION

---

If, while accessing a stream file, you seek beyond the end-of-file and then write to the file, the `fseek` function creates a hole by filling the skipped bytes with zeros.

In general, for record files, `fseek` should only be directed to an absolute position that was returned by a previous valid call to `ftell`, or to the beginning or end of a file. If a call to `fseek` does not satisfy these conditions, the results are unpredictable.

---

See also `open`, `creat`, `dup`, `dup2`, and `lseek` in this section.

**Return Values**

|    |                             |
|----|-----------------------------|
| 0  | Indicates successful seeks. |
| -1 | Indicates improper seeks.   |

## fseeko

---

## fseeko

Positions the file to the specified byte offset in the file. Equivalent to `fseek`.

### Format

```
#include <stdio.h>

int fseeko (FILE *file_ptr, off_t offset, int direction);
```

### Arguments

**file\_ptr**

A file pointer.

**offset**

The offset, specified in bytes. The `off_t` data type is either a 32-bit integer or 64-bit integer. The 64-bit interface allows for file sizes greater than 2 gigabytes, and can be selected at compile time by defining the `_LARGEFILE` feature-test macro as follows:

```
CC/DEFINE=_LARGEFILE
```

**direction**

An integer indicating the position to which the *offset* is added to calculate the new position. The new position is the beginning of the file if *direction* is `SEEK_SET`, the current value of the file position indicator if *direction* is `SEEK_CUR`, or end-of-file if *direction* is `SEEK_END`.

### Description

The `fseeko` function is identical to the `fseek` function, except that the *offset* argument is of type `off_t` instead of `long int`.

---

## fsetpos

Sets the file position indicator for a given file.

### Format

```
#include <stdio.h>

int fsetpos (FILE *stream, const fpos_t *pos);
```

### Arguments

**stream**

A file pointer.

**pos**

A pointer to an implementation-defined structure. The `fgetpos` function fills this structure with information that can be used on subsequent calls to `fsetpos`.

### Description

Call the `fgetpos` function before using the `fsetpos` function.

### Return Values

|    |                     |
|----|---------------------|
| 0  | Indicates success.  |
| −1 | Indicates an error. |

## fstat

---

## fstat

Accesses information about the file specified by the file descriptor.

### Format

```
#include <stat.h>

int fstat (int file_desc, struct stat *buffer);
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `fstat` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Arguments

#### **file\_desc**

A file descriptor.

#### **buffer**

A pointer to a structure of type `stat_t`, which is defined in the `<stat.h>` header file. The argument receives information about that particular file. The members of the structure pointed to by *buffer* are as follows:

| Member                  | Type                  | Definition                                                   |
|-------------------------|-----------------------|--------------------------------------------------------------|
| <code>st_dev</code>     | <code>dev_t</code>    | Pointer to a physical device name                            |
| <code>st_ino[3]</code>  | <code>ino_t</code>    | Three words to receive the file ID                           |
| <code>st_mode</code>    | <code>mode_t</code>   | File “mode” ( <code>prot</code> , <code>dir</code> , . . . ) |
| <code>st_nlink</code>   | <code>nlink_t</code>  | For UNIX system compatibility only                           |
| <code>st_uid</code>     | <code>uid_t</code>    | Owner user ID                                                |
| <code>st_gid</code>     | <code>gid_t</code>    | Group member: from <code>st_uid</code>                       |
| <code>st_rdev</code>    | <code>dev_t</code>    | UNIX system compatibility – always 0                         |
| <code>st_size</code>    | <code>off_t</code>    | File size, in bytes                                          |
| <code>st_atime</code>   | <code>time_t</code>   | File access time; always the same as <code>st_mtime</code>   |
| <code>st_mtime</code>   | <code>time_t</code>   | Last modification time                                       |
| <code>st_ctime</code>   | <code>time_t</code>   | File creation time                                           |
| <code>st_fab_rfm</code> | <code>char</code>     | Record format                                                |
| <code>st_fab_rat</code> | <code>char</code>     | Record attributes                                            |
| <code>st_fab_fsz</code> | <code>char</code>     | Fixed header size                                            |
| <code>st_fab_mrs</code> | <code>unsigned</code> | Record size                                                  |

The types `dev_t`, `ino_t`, `off_t`, `mode_t`, `nlink_t`, `uid_t`, `gid_t`, and `time_t`, are defined in the `<stat.h>` header file. However, when compiling for compatibility (`/DEFINE=_DECC_V4_SOURCE`), only `dev_t`, `ino_t`, and `off_t` are defined.



The `off_t` data type is either a 32-bit integer or 64-bit integer. The 64-bit interface allows for file sizes greater than 2 gigabytes, and can be selected at compile time by defining the `_LARGEFILE` feature-test macro as follows:

```
CC/DEFINE=_LARGEFILE
```

As of OpenVMS Version 7.0, times are given in seconds since the Epoch (00:00:00 GMT, January 1, 1970).

The `st_mode` structure member is the status information mode and is defined in the `<stat.h>` header file. The `st_mode` bits follow:

| Bits    | Constant | Definition                       |
|---------|----------|----------------------------------|
| 0170000 | S_IFMT   | Type of file                     |
| 0040000 | S_IFDIR  | Directory                        |
| 0020000 | S_IFCHR  | Character special                |
| 0060000 | S_IFBLK  | Block special                    |
| 0100000 | S_IFREG  | Regular                          |
| 0030000 | S_IFMPC  | Multiplexed char special         |
| 0070000 | S_IFMPB  | Multiplexed block special        |
| 0004000 | S_ISUID  | Set user ID on execution         |
| 0002000 | S_ISGID  | Set group ID on execution        |
| 0001000 | S_ISVTX  | Save swapped text even after use |
| 0000400 | S_IREAD  | Read permission, owner           |
| 0000200 | S_IWRITE | Write permission, owner          |
| 0000100 | S_IEXEC  | Execute/search permission, owner |

## Description

The `fstat` function does not work on remote network files.

---

### Note *(Alpha only)*

---

On OpenVMS Alpha systems, the `stat`, `fstat`, `utime`, and `utimes` functions have been enhanced to take advantage of the new file-system support for POSIX-compliant file timestamps.

This support is available only on ODS-5 devices on OpenVMS Alpha systems beginning with a version of OpenVMS Alpha after Version 7.3.

Before this change, the `stat` and `fstat` functions were setting the values of the `st_ctime`, `st_mtime`, and `st_atime` fields based on the following file attributes:

- `st_ctime` - `ATR$C_CREDATE` (file creation time)
- `st_mtime` - `ATR$C_REVDATE` (file revision time)
- `st_atime` - was always set to `st_mtime` because no support for file access time was available

Also, for the file-modification time, `utime` and `utimes` were modifying the `ATR$C_REVDATE` file attribute, and ignoring the file-access-time argument.

## fstat

After the change, for a file on an ODS-5 device, the `stat` and `fstat` functions set the values of the `st_ctime`, `st_mtime`, and `st_atime` fields based on the following new file attributes:

`st_ctime` - `ATR$C_ATTDATE` (last attribute modification time)  
`st_mtime` - `ATR$C_MODDATE` (last data modification time)  
`st_atime` - `ATR$C_ACCDATE` (last access time)

If `ATR$C_ACCDATE` is zero, as on an ODS-2 device, the `stat` and `fstat` functions set `st_atime` to `st_mtime`.

For the file-modification time, the `utime` and `utimes` functions modify both the `ATR$C_REVDATE` and `ATR$C_MODDATE` file attributes. For the file-access time, these functions modify the `ATR$C_ACCDATE` file attribute. Setting the `ATR$C_MODDATE` and `ATR$C_ACCDATE` file attributes on an ODS-2 device has no effect.

For compatibility, the old behavior of `stat`, `fstat`, `utime`, and `utimes` remains the default, regardless of the kind of device.

The new behavior must be explicitly enabled at run time by defining the `DECC$EFS_FILE_TIMESTAMPS` logical name to "ENABLE" before invoking the application. Setting this logical does not affect the behavior of `stat`, `fstat`, `utime` and `utimes` for files on an ODS-2 device.

---

## Return Values

|    |                                                       |
|----|-------------------------------------------------------|
| 0  | Indicates successful completion.                      |
| -1 | Indicates an error other than a protection violation. |
| -2 | Indicates a protection violation.                     |

---

## fsync

Flushes data all the way to the disk.

### Format

```
#include <unistd.h>

int fsync (int fd);
```

### Argument

**fd**  
A file descriptor corresponding to an open file.

### Description

The `fsync` function behaves much like the `fflush` function. The primary difference between the two is that `fsync` flushes data all the way to the disk while `fflush` flushes data only as far as the underlying RMS buffers. Also, with `fflush`, you can flush all buffers at once; with `fsync` you cannot.

### Return Values

|    |                                  |
|----|----------------------------------|
| 0  | Indicates successful completion. |
| −1 | Indicates an error.              |

## ftell

---

## ftell

Returns the current byte offset to the specified stream file.

### Format

```
#include <stdio.h>

long int ftell (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A file pointer.

### Description

The `ftell` function measures the byte offset from the beginning of the file.

For variable-length files, VFC files, or any file with carriage-control attributes, if the file is opened in record mode, then `ftell` returns the starting position of the current record, not the current byte offset.

When using record files, the `ftell` function ignores any characters that have been pushed back using either `ungetc` or `ungetwc`. This behavior does not occur if stream files are being used.

For a portable way to measure the exact offset for any type of file, see the `fgetpos` function.

### Return Values

|                  |                     |
|------------------|---------------------|
| <code>n</code>   | The current offset. |
| <code>EOF</code> | Indicates an error. |

---

## ftello

Returns the current byte offset to the specified stream file. This function is equivalent to `ftell`.

### Format

```
#include <stdio.h>

off_t ftello (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A file pointer.

### Description

The `ftello` function is identical to the `ftell` function, except that the return value is of type `off_t` instead of `long int`.

The `off_t` data type is either a 64-bit integer or a 32-bit integer. The 64-bit interface allows for file sizes greater than 2 gigabytes, and can be selected at compile time by defining the `_LARGEFILE` feature-test macro as follows:

```
CC/DEFINE=_LARGEFILE
```

## ftime

---

## ftime

Returns the elapsed time since 00:00:00, January 1, 1970, in the structure pointed at by *timeptr*.

### Format

```
#include <timeb.h>

int ftime (struct timeb *timeptr);
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `ftime` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Argument

#### **timeptr**

A pointer to the structure `timeb_t`.

### Description

The typedef `timeb_t` refers to the following structure defined in the `<timeb.h>` header file:

```
typedef struct timeb
{
 time_t time;
 unsigned short millitm;
 short timezone;
 short dstflag;
};
```

The member `time` gives the time in seconds.

The member `millitm` gives the fractional time in milliseconds.

After a call to `ftime`, the `timezone` and `dstflag` members of the `timeb` structure have the values of the global variables `timezone` and `dstflag`, respectively. See the description of the `tzset` function for `timezone` and `dstflag` global variables.

### Return Values

|    |                                                                                                                                                                                                                                                                                                                                     |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Successful execution. The <code>timeb_t</code> structure is filled in.                                                                                                                                                                                                                                                              |
| -1 | Indicates an error. Failure might indicate that the system's time-differential factor (that is, the difference between the system time and UTC time) is not set correctly.<br>If the value of the <code>SYS\$TIMEZONE_DIFFERENTIAL</code> logical is wrong, the function fails with <code>errno</code> set to <code>EINVAL</code> . |

---

## ftruncate

Truncates a file to a specified length.

### Format

```
#include <unistd.h>

int ftruncate (int filedes, off_t length);
```

### Arguments

**filedes**

The descriptor of a file that must be open for writing.

**length**

The new length of the file in bytes. The `off_t` data type is either a 32-bit integer or 64-bit integer. The 64-bit interface allows for file sizes greater than 2 gigabytes, and can be selected at compile time by defining the `_LARGEFILE` feature-test macro as follows:

```
CC/DEFINE=_LARGEFILE
```

### Description

The `ftruncate` function truncates a file at the specified position. For record files, the position must be a record boundary. Also, the files must be local, regular files.

If the file was previously larger than *length*, extra data is lost. If the file was previously shorter than *length*, bytes between the old and new lengths are read as zeros.

### Return Values

|    |                                                                     |
|----|---------------------------------------------------------------------|
| 0  | Indicates success.                                                  |
| -1 | An error occurred; <code>errno</code> is set to indicate the error. |

---

**ftw**

Walks a file tree.

**Format**

```
#include <ftw.h>
```

```
int ftw (const char *path, int(*function)(const char *, const struct stat *, int), int depth);
```

**Arguments****path**

The directory hierarchy to be searched.

**function**

The function to be invoked for each file in the directory hierarchy.

**depth**

The maximum number of directory streams or file descriptors, or both, available for use by ftw. This argument should be in the range of 1 to OPEN\_MAX.

**Description**

The ftw function recursively searches the directory hierarchy that descends from the directory specified by the *path* argument.

For each file in the hierarchy, ftw calls the function specified by the *function* argument, passes it a pointer to a null-terminated character string containing the name of the file, a pointer to a stat structure containing information about the file, and an integer.

The integer identifies the file type. Possible values, defined in <ftw.h> are:

|         |                                                          |
|---------|----------------------------------------------------------|
| FTW_F   | Regular file.                                            |
| FTW_D   | Directory.                                               |
| FTW_DNR | Directory that cannot be read.                           |
| FTW_NS  | A file on which stat could not successfully be executed. |

If the integer is FTW\_DNR, then the files and subdirectories contained in that directory are not processed.

If the integer is FTW\_NS, then the stat structure contents are meaningless. For example, a file in a directory for which you have read permission but not execute (search) permission can cause the *function* argument to pass FTW\_NS.

The ftw function finishes processing a directory before processing any of its files or subdirectories.

The ftw function continues the search until:

- The directory hierarchy specified by the *path* argument is completed.
- An invocation of the function specified by the *function* argument returns a nonzero value.
- An error (such as an I/O error) is detected within the ftw function.

Because the ftw function is recursive, it is possible for it to terminate with a memory fault because of stack overflow when applied to very deep file structures.



The `ftw` function uses the `malloc` function to allocate dynamic storage during its operation. If `ftw` is forcibly terminated, as with a call to `longjmp` from the function pointed to by the *function* argument, `ftw` has no chance to free that storage. It remains allocated.

A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function specified by the *function* argument return a nonzero value the next time it is called.

---

#### Note

---

The `ftw` function is reentrant; make sure that the function supplied as argument function is also reentrant.

---

See also `malloc`, `longjump`, `lstat`, and `stat` in this section.

## Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| x  | Indicates that the function specified by the <i>function</i> argument stops its search, and returns the value that was returned by the function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| -1 | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"> <li>• <code>EACCES</code> – Search permission is denied for any component of the <i>path</i> argument or read permission is denied for the <i>path</i> argument.</li> <li>• <code>ENAMETOOLONG</code> – The length of the path string exceeds <code>PATH_MAX</code>, or a pathname component is longer than <code>NAME_MAX</code> while <code>[_POSIX_NO_TRUNC]</code> is in effect.</li> <li>• <code>ENOENT</code> – The <i>path</i> argument points to the name of a file that does not exist or points to an empty string.</li> <li>• <code>ENOMEM</code> – There is insufficient memory for this operation.</li> </ul> |

Also, if the function pointed to by the *function* argument encounters an error, `errno` can be set accordingly.

## **fwait**

---

### **fwait**

Waits for I/O on a specific file to complete.

#### **Format**

```
#include <stdio.h>
int fwait (FILE *fp);
```

#### **Argument**

**fp**  
A file pointer corresponding to an open file.

#### **Description**

The `fwait` function is used primarily to wait for completion of pending asynchronous I/O.

#### **Return Values**

|    |                                  |
|----|----------------------------------|
| 0  | Indicates successful completion. |
| −1 | Indicates an error.              |

---

## fwide

Determines and sets the orientation of a stream.

### Format

```
#include <wchar.h>

int fwide (FILE *stream, int mode);
```

### Arguments

**stream**

A file pointer.

**mode**

A value that specifies the desired orientation of the stream.

### Description

The `fwide` function determines the orientation of the stream pointed to by *stream* and sets the orientation of a non-oriented stream according to the *mode* argument in the following way:

| If the <i>mode</i> argument is: | Then the <i>fwide</i> function:               |
|---------------------------------|-----------------------------------------------|
| greater than zero               | makes the stream wide-oriented.               |
| less than zero                  | makes the stream byte-oriented.               |
| zero                            | does not alter the orientation of the stream. |

If the orientation of the stream has already been set, `fwide` does not alter it. Because no error status is defined for `fwide`, the calling application should check `errno` if `fwide` returns a 0.

### Return Values

|     |                                                                                                                       |
|-----|-----------------------------------------------------------------------------------------------------------------------|
| > 0 | After the call, the stream is wide-oriented.                                                                          |
| < 0 | After the call, the stream is byte-oriented.                                                                          |
| 0   | After the call, the stream has no orientation or a stream argument is invalid; the function sets <code>errno</code> . |

## fwprintf

---

## fwprintf

Writes output to the stream under control of the wide-character format string.

### Format

```
#include <wchar.h>

int fwprintf (FILE *stream, const wchar_t *format, ...);
```

### Arguments

#### **stream**

A file pointer.

#### **format**

A pointer to a wide-character string containing the format specifications. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

...

Optional expressions whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, the output sources can be omitted. Otherwise, the function calls must have exactly as many output sources as there are conversion specifications, and the conversion specifications must match the types of the output sources.

Conversion specifications are matched to output sources in left-to-right order. Any excess output sources are ignored.

### Description

The `fwprintf` function writes output to the stream pointed to by *stream* under control of the wide-character string pointed to by *format*, which specifies how to convert subsequent arguments to output. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated, but are otherwise ignored. The `fwprintf` function returns when it encounters the end of the format string.

The *format* argument is composed of zero or more directives that include:

- Ordinary wide characters (not the percent sign (%))
- Conversion specifications

## Return Values

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n              | The number of wide characters written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Negative value | <p>Indicates an error. The function sets <code>errno</code> to one of the following:</p> <ul style="list-style-type: none"> <li>• EILSEQ – Invalid character detected.</li> <li>• EINVAL – Insufficient arguments.</li> <li>• ENOMEM – Not enough memory available for conversion.</li> <li>• ERANGE – Floating-point calculations overflow.</li> <li>• EVMSERR – Nontranslatable VMS error. <code>vaxc\$errno</code> contains the VMS error code. This might indicate that conversion to a numeric value failed because of overflow.</li> </ul> <p>The function can also set <code>errno</code> to the following as a result of errors returned from the I/O subsystem:</p> <ul style="list-style-type: none"> <li>• EBADF – The file descriptor is not valid.</li> <li>• EIO – I/O error.</li> <li>• ENOSPC – No free space on the device containing the file.</li> <li>• ENXIO – Device does not exist.</li> <li>• EPIPE – Broken pipe.</li> <li>• ESPIPE – Illegal seek in a file opened for append.</li> <li>• EVMSERR – Nontranslatable VMS error. <code>vaxc\$errno</code> contains the VMS error code. This indicates that an I/O error occurred for which there is no equivalent C error code.</li> </ul> |

## Example

The following example shows how to print a date and time in the form "Sunday, July 3, 10:02", followed by  $\pi$  to five decimal places:

```
#include <math.h>
#include <stdio.h>
#include <wchar.h>
/* ... */
wchar_t *weekday, *month; /* pointers to wide-character strings */
int day, hours, min;
fwprintf(stdout, L"%ls, %ls %d, %.2d:%.2d\n",
 weekday, month, day, hour, min);
fwprintf(stdout, L"pi = %.5f\n", 4 * atan(1.0));
```

## **fwrite**

---

## **fwrite**

Writes a specified number of items to the file.

### **Format**

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

```
size_t fwrite (const void *ptr, size_t size_of_item, size_t number_items, FILE *file_ptr);
```

### **Arguments**

#### **ptr**

A pointer to the memory location from which information is being written. The type of the object pointed to is determined by the type of the item being written.

#### **size\_of\_item**

The size, in bytes, of the items being written.

#### **number\_items**

The number of items to be written.

#### **file\_ptr**

A file pointer that indicates the file to which the items are being written.

### **Description**

The type `size_t` is defined in the header file `<stdio.h>` as follows:

```
typedef unsigned int size_t
```

The writing begins at the current location in the file. The items are written from storage beginning at the location given by the first argument. You must also specify the size of an item, in bytes.

If the file pointed to by *file\_ptr* is a record file, the `fwrite` function outputs at least *number\_items* records, each of length *size\_of\_item*.

### **Return Value**

x

The number of items written. The number of records written depends upon the maximum record size of the file.

---

## fwscanf

Reads input from the stream under control of the wide-character format string.

### Format

```
#include <wchar.h>

int fwscanf (FILE *stream, const wchar_t *format, ...);
```

### Arguments

**stream**

A file pointer.

**format**

A pointer to a wide-character string containing the format specification. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

...

Optional expressions whose results correspond to conversion specifications given in the format specification. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

If no conversion specifications are given, you can omit the input pointers. Otherwise, the function calls must have exactly as many input pointers as there are conversion specifications, and the conversion specifications must match the types of the input pointers.

Conversion specifications are matched to input sources in left-to-right order. Excess input pointers, if any, are ignored.

### Description

The `fwscanf` function reads input from the stream pointed to by *stream* under the control of the wide-character string pointed to by *format*. If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated, but otherwise ignored.

The format is composed of zero or more directives that include:

- One or more white-space wide characters.
- An ordinary wide character (neither a percent ( `%` )) nor a white-space wide character).
- Conversion specifications.

Each conversion specification is introduced by the wide character `%`.

If the stream pointed to by the stream argument has no orientation, `fwscanf` makes the stream wide-oriented.

## **fwscanf**

### **Return Values**

|     |                                                                                                                                 |
|-----|---------------------------------------------------------------------------------------------------------------------------------|
| n   | The number of input items assigned, sometimes fewer than provided for, or even zero, in the event of an early matching failure. |
| EOF | Indicates an error; input failure occurs before any conversion.                                                                 |



---

## gcv

Converts its argument to a null-terminated string of ASCII digits and returns the address of the string.

### Format

```
#include <stdlib.h>

char *gcv (double value, int ndigit, char *buffer);
```

### Function Variants

The gcv function has variants named `_gcv32` and `_gcv64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**value**

An object of type double that is converted to a null-terminated string of ASCII digits.

**ndigit**

The number of ASCII digits to use in the converted string. If *ndigit* is less than 6, the value of 6 is used.

**buffer**

A storage location to hold the converted string.

### Description

The gcv function places the converted string in a buffer and returns the address of the buffer. If possible, gcv produces *ndigit* significant digits in F-format, or if not possible, in E-format. Trailing zeros are suppressed.

The ecvt, fcvt, and gcv functions represent the following special values specified in the IEEE Standard for floating-point arithmetic:

| Value          | Representation |
|----------------|----------------|
| Quiet NaN      | NaNQ           |
| Signalling NaN | NaNS           |
| +Infinity      | Infinity       |
| –Infinity      | –Infinity      |

The sign associated with each of these values is stored into the *sign* argument. In IEEE floating-point representation, a value of 0 (zero) can be positive or negative, as set by the *sign* argument.

See also fcvt and ecvt in this section.

**gcv**

**Return Value**

x

The address of the buffer.

---

## getc

The `getc` macro returns the next character from a specified file.

### Format

```
#include <stdio.h>

int getc (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A pointer to the file to be accessed.

### Description

Since `getc` is a macro, a file pointer argument with side effects (for example, `getc (*f++)`) might be evaluated incorrectly. In such a case, use the `fgetc` function instead. See the `fgetc` function in this section.

### Return Values

|                  |                                        |
|------------------|----------------------------------------|
| <code>n</code>   | The returned character.                |
| <code>EOF</code> | Indicates the end-of-file or an error. |

## [w]getch

---

## [w]getch

Get a character from the terminal screen and echo it on the specified window. The getch function echoes the character on the stdscr window.

### Format

```
#include <curses.h>

char getch();

char wgetch (WINDOW *win);
```

### Argument

**win**  
A pointer to the window.

### Description

The getch and wgetch functions refresh the specified window before fetching a character. For more information, see the `scrollok` function in this section.

### Return Values

|     |                                                                |
|-----|----------------------------------------------------------------|
| x   | The returned character.                                        |
| ERR | Indicates that the function makes the screen scroll illegally. |

---

## **getchar**

Reads a single character from the standard input (stdin).

### **Format**

```
#include <stdio.h>

int getchar (void);
```

### **Description**

The `getchar` function is identical to `fgetc(stdin)`.

### **Return Values**

|                  |                                                                              |
|------------------|------------------------------------------------------------------------------|
| <code>x</code>   | The next character from <code>stdin</code> , converted to <code>int</code> . |
| <code>EOF</code> | Indicates the end-of-file or an error.                                       |

## getclock

---

## getclock

Gets the current value of the systemwide clock.

### Format

```
#include <timers.h>

int getclock (int clktyp, struct timespec *tp);
```

### Arguments

#### **clktyp**

The type of systemwide clock.

#### **tp**

Pointer to a timespec structure space where the current value of the systemwide clock is stored.

### Description

The getclock function sets the current value of the clock specified by *clktyp* into the location pointed to by *tp*.

The *clktyp* argument is given as a symbolic constant name, as defined in the <timers.h> header file. Only the TIMEOFDAY symbolic constant, which specifies the normal time-of-day clock to access for systemwide time, is supported.

For the clock specified by TIMEOFDAY, the value returned by this function is the elapsed time since the Epoch. The Epoch is referenced to 00:00:00 UTC (Coordinated Universal Time) 1 Jan 1970.

The getclock function returns a timespec structure, which is defined in the <timers.h> header file as follows:

```
struct timespec {
 unsigned long tv_sec /* Elapsed time in seconds since the Epoch*/
 long tv_nsec /* Elapsed time as a fraction of a second */
 /* since the Epoch (in nanoseconds) */
};
```

### Return Values

0

Indicates success.

−1

Indicates an error; errno is set to one of the following values:

- EINVAL – The *clktyp* argument does not specify a known systemwide clock.  
Or, the value of SYS\$TIMEZONE\_DIFFERENTIAL logical is wrong.
- EIO – An error occurred when the systemwide clock specified by the *clktyp* argument was accessed.

---

## getcwd

Returns a pointer to the file specification for the current working directory.

### Format

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

```
char *getcwd (char *buffer, size_t size); (ISO POSIX-1)
```

```
char *getcwd (char *buffer, unsigned int size, . . .); (COMPAQ C Extension)
```

### Function Variants

The `getcwd` function has variants named `_getcwd32` and `_getcwd64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **buffer**

Pointer to a character string large enough to hold the directory specification.

If *buffer* is a NULL pointer, `getcwd` obtains *size* bytes of space using `malloc`. In this case, you can use the pointer returned by `getcwd` as the argument in a subsequent call to `free`.

#### **size**

The length of the directory specification to be returned.

. . .

An optional argument that can be either 1 or 0. If you specify 1, the `getcwd` function returns the directory specification in OpenVMS format. If you specify 0, `getcwd` returns the directory specification (pathname) in UNIX style format. If you do not specify this argument, `getcwd` returns the file name according to your current command-language interpreter. For more information about UNIX style directory specifications, see Section 1.4.3.

### Return Values

|      |                                      |
|------|--------------------------------------|
| x    | A pointer to the file specification. |
| NULL | Indicates an error.                  |

## getdtablesize

---

## getdtablesize

Gets the total number of file descriptors that a process can have open simultaneously.

### Format

```
#include <unistd.h>

int getdtablesize (void);
```

### Description

The `getdtablesize` function returns the total number of file descriptors that a process can have open simultaneously. Each process is limited to a fixed number of open file descriptors.

The number of file descriptors that a process can have open is the minimum of the following:

- *Compaq C* RTL open file limit—65535 on OpenVMS Alpha; 2048 on OpenVMS VAX.
- SYSGEN CHANNELCNT parameter—permanent I/O channel count.
- Process open file quota FILLM parameter—number of open files that can be opened by a process at one time.

### Return Values

|    |                                                                             |
|----|-----------------------------------------------------------------------------|
| x  | The number of file descriptors that a process can have open simultaneously. |
| −1 | Indicates an error.                                                         |



---

## getegid

With POSIX IDs disabled, this function is equivalent to `getgid` and returns the group number from the user identification code (UIC).

With POSIX IDs enabled, returns the effective group ID.

### Format

```
#include <unistd.h>

gid_t getegid (void);
```

### Description

The `getegid` function can be used with POSIX style identifiers or with UIC-based identifiers.

---

#### Note

---

Currently, POSIX style IDs are supported only by some OpenVMS versions done for specific government agencies, but will be integrated into future OpenVMS releases. OpenVMS Version 7.3-1 does not support POSIX style IDs, but it does support 32-bit identifiers.

---

With POSIX style IDs disabled (the default), the `getegid` and `getgid` functions are equivalent and return the group number from the current UIC. For example, if the UIC is [313,031], 313 is the group number.

With POSIX style IDs enabled, `getegid` returns the effective group ID of the calling process, and `getgid` returns the real group ID of the calling process. The real group ID is specified at login time. The effective group ID is more transient, and determines additional access permission during execution of a *set-group-ID* process. It is for such processes that the `getgid` function is most useful.

See also `geteuid` and `getuid` in this section.

### Return Value

|   |                                                                                                    |
|---|----------------------------------------------------------------------------------------------------|
| x | The effective group ID (POSIX IDs enabled), or the group number from the UIC (POSIX IDs disabled). |
|---|----------------------------------------------------------------------------------------------------|

---

## getenv

Searches the environment array for the current process and returns the value associated with a specified environment name.

### Format

```
#include <stdlib.h>

char *getenv (const char *name);
```

### Argument

#### **name**

One of the following values:

- HOME—Your login directory
- TERM—The type of terminal being used
- PATH—The default device and directory
- USER—The name of the user who initiated the process
- Logical name or CLI symbolic name
- An environment variable set with `setenv` or `putenv`

The case of the specified *name* is important.

### Description

In certain situations, the `getenv` function attempts to perform a logical name translation on the user-specified argument:

1. If the argument to `getenv` does not match any of the environment strings present in your environment array, `getenv` attempts to translate your argument as a logical name by searching the logical name tables indicated by the `LN$FILE_DEV` logical, as is done for file processing.

`getenv` first does a case-sensitive lookup. If that fails, it does a case-insensitive lookup. In most instances, logical names are defined in uppercase, but `getenv` can also find logical names that include lowercase letters.

`getenv` does not perform iterative logical name translation.

2. If the logical name is a search list with multiple equivalence values, the returned value points to the first equivalence value. For example:

```
$ DEFINE A B,C
```

```
ptr = getenv("A");
```

A returns a pointer to "B".

3. If no logical name exists, `getenv` attempts to translate the argument string as a command-language interpreter (CLI) symbol. If it succeeds, it returns the translated symbol text. If it fails, the return value is `NULL`.

`getenv` does not perform iterative CLI translation.

If your CLI is the DEC/Shell, the function does not attempt a logical name translation since Shell environment symbols are implemented as DCL symbols.

---

**Note**

---

In OpenVMS Version 7.1, a cache of VMS environment variables (that is, logical names and DCL symbols) has been added to the `getenv` function to avoid the library making repeated calls to translate a logical name or to obtain the value of a DCL symbol. By default, the cache is disabled. If your application does not need to track changes in OpenVMS environment variables that can occur during its execution, the cache can be enabled by setting the `DECC$ENABLE_GETENV_CACHE` logical before invoking the application (any equivalence string).

---

## Return Values

|                   |                                                                                                      |
|-------------------|------------------------------------------------------------------------------------------------------|
| <code>x</code>    | Pointer to an array containing the translated symbol. An equivalence name is returned at index zero. |
| <code>NULL</code> | Indicates that the translation failed.                                                               |

## geteuid

---

## geteuid

With POSIX IDs disabled, this function is equivalent to `getuid` and returns the member number (in OpenVMS terms) from the user identification code (UIC).

With POSIX IDs enabled, returns the effective user ID.

### Format

```
#include <unistd.h>

uid_t geteuid (void);
```

### Description

The `geteuid` function can be used with POSIX style identifiers or with UIC-based identifiers.

---

#### Note

---

Currently, POSIX style IDs are supported only by some OpenVMS versions done for specific government agencies, but will be integrated into future OpenVMS releases. OpenVMS Version 7.3-1 does not support POSIX style IDs, but it does support 32-bit identifiers.

---

With POSIX style IDs disabled (the default), the `geteuid` and `getuid` functions are equivalent and return the member number from the current UIC as follows:

- For programs compiled with the `_VMS_V6_SOURCE` feature-test macro or programs that do not include the `<unistd.h>` header file, the `getuid` and `geteuid` functions return the member number of the OpenVMS UIC. For example, if the UIC is [313,31], then the member number, 31, is returned.
- For programs compiled without the `_VMS_V6_SOURCE` feature-test macro that do include the `<unistd.h>` header file, the full UIC is returned. For example, if the UIC is [313, 31] then 20512799 (31 + 313 \* 65536) is returned.

With POSIX style IDs enabled, `geteuid` returns the effective user ID of the calling process, and `getuid` returns the real user ID of the calling process.

See also `getegid` and `getgid` in this section.

### Return Value

|   |                                                                                                                            |
|---|----------------------------------------------------------------------------------------------------------------------------|
| x | The effective user ID (POSIX IDs enabled), or the member number from the current UIC or the full UIC (POSIX IDs disabled). |
|---|----------------------------------------------------------------------------------------------------------------------------|

---

## getgid

With POSIX IDs disabled, this function is equivalent to `getegid` and returns the group number from the user identification code (UIC).

With POSIX IDs enabled, returns the real group ID.

### Format

```
#include <unistd.h>

gid_t getgid (void);
```

### Description

The `getgid` function can be used with POSIX style identifiers or with UIC-based identifiers.

---

#### Note

Currently, POSIX style IDs are supported only by some OpenVMS versions done for specific government agencies, but will be integrated into future OpenVMS releases. OpenVMS Version 7.3-1 does not support POSIX style IDs, but it does support 32-bit identifiers.

---

With POSIX style IDs disabled (the default), the `getegid` and `getgid` functions are equivalent and return the group number from the current UIC. For example, if the UIC is [313,031], 313 is the group number.

With POSIX style IDs enabled, `getegid` returns the effective group ID of the calling process, and `getgid` returns the real group ID of the calling process. The real group ID is specified at login time. The effective group ID is more transient, and determines additional access permission during execution of a *set-group-ID* process. It is for such processes that the `getgid` function is most useful.

See also `geteuid` and `getuid` in this section.

### Return Value

|   |                                                                                                       |
|---|-------------------------------------------------------------------------------------------------------|
| x | The real group ID (POSIX IDs enabled), or the group number from the current UIC (POSIX IDs disabled). |
|---|-------------------------------------------------------------------------------------------------------|

---

## getitimer

Returns the value of interval timers.

### Format

```
#include <time.h>

int getitimer (int which, struct itimerval *value);
```

### Arguments

#### which

The type of interval timer. The *Compaq C* RTL supports only ITIMER\_REAL.

#### value

Pointer to an `itimerval` structure whose members specify a timer interval and the time left to the end of the interval.

### Description

The `getitimer` function returns the current value for the timer specified by the *which* argument in the structure pointed to by *value*.

A timer value is defined by the `itimerval` structure:

```
struct itimerval {
 struct timeval it_interval;
 struct timeval it_value;
};
```

The following table lists the values for the `itimerval` structure members:

| itimerval Member Value | Meaning                                                                    |
|------------------------|----------------------------------------------------------------------------|
| it_interval = 0        | Disables a timer after its next expiration<br>Assumes it_value is nonzero. |
| it_interval = nonzero  | Specifies a value used in reloading it_value<br>when the timer expires.    |
| it_value = 0           | Disables a timer.                                                          |
| it_value = nonzero     | Indicates the time to the next timer expiration.                           |

Time values smaller than the resolution of the system clock are rounded up to this resolution.

The *Compaq C* RTL provides each process with one interval timer, defined in the `<time.h>` header file as ITIMER\_REAL. This timer decrements in real time and delivers a SIGALRM signal when the timer expires.

**Return Values**

|    |                                                                                                                                          |
|----|------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                       |
| -1 | Indicates an error; <code>errno</code> is set to <code>EINVAL</code> (The value argument specified a time that was too large to handle.) |

## getlogin

---

## getlogin

Gets the login name.

### Format

```
#include <unistd.h>
char *getlogin (void);
```

### Description

The `getlogin` function returns the login name of the user associated with the current session.

### Return Values

|      |                                                           |
|------|-----------------------------------------------------------|
| x    | A pointer to a null-terminated string in a static buffer. |
| NULL | Indicates an error. Login name is not set.                |



---

## getname

Returns the file specification associated with a file descriptor.

### Format

```
#include <unixio.h>

char *getname (int file_desc, char *buffer, ...);
```

### Function Variants

The `getname` function has variants named `_getname32` and `_getname64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**file\_desc**

A file descriptor.

**buffer**

A pointer to a character string that is large enough to hold the file specification.

...

An optional argument that can be either 1 or 0. If you specify 1, the `getname` function returns the file specification in OpenVMS format. If you specify 0, the `getname` function returns the file specification in UNIX style format. If you do not specify this argument, the `getname` function returns the file name according to your current command-language interpreter (CLI). For more information about UNIX style file specifications, see Section 1.4.3.

### Description

The `getname` function places the file specification into the area pointed to by *buffer* and returns that address. The area pointed to by *buffer* should be an array large enough to contain a fully qualified file specification (the maximum length is 256 characters).

### Return Values

|   |                                                   |
|---|---------------------------------------------------|
| x | The address passed in the <i>buffer</i> argument. |
| 0 | Indicates an error.                               |

## getopt

---

## getopt

A command-line parser that can be used by applications that follow UNIX command-line conventions.

### Format

```
#include <unistd.h> (X/Open, POSIX-1)
#include <stdio.h> (X/Open, POSIX-2)

int getopt (int argc, char * const argv[], const char *optstring);

extern char *optarg;

extern int optind, opterr, optopt;
```

### Arguments

#### **argc**

The argument count as passed to main.

#### **argv**

The argument array as passed to main.

#### **optstring**

A string of recognized option characters. If a character is followed by a colon, the option takes an argument.

### Description

The variable `optind` is the index of the next element of the `argv` vector to be processed. It is initialized to 1 by the system, and it is updated by `getopt` when it finishes with each element of `argv`. When an element of `argv` contains multiple option characters, it is unspecified how `getopt` determines which options have already been processed.

The `getopt` function returns the next option character (if one is found) from `argv` that matches a character in `optstring`, if there is one that matches. If the option takes an argument, `getopt` sets the variable `optarg` to point to the option-argument as follows:

- If the option was the last character in the string pointed to by an element of `argv`, then `optarg` contains the next element of `argv`, and `optind` is incremented by 2. If the resulting value of `optind` is not less than `argc`, `getopt` returns an error, indicating a missing option-argument.
- Otherwise, `optarg` points to the string following the option character in that element of `argv`, and `optind` is incremented by 1.

If one of the following is true, `getopt` returns `-1` without changing `optind`:

- `argv[optind]` is a NULL pointer
- `*argv[optind]` is not the character `-`
- `argv[optind]` points to the string `"-"`

If `argv[optind]` points to the string `"--"` `getopt` returns `-1` after incrementing `optind`.

If getopt encounters an option character not contained in *optstring*, the question-mark character (?) is returned.

If getopt detects a missing argument, the colon character (:) is returned if the first character of *optstring* is a colon; otherwise a question-mark character is returned.

In either of the previous two cases, getopt sets the variable *optopt* to the option character that caused the error. If the application has not set the variable *opterr* to 0 and the first character of *optstring* is not a colon, getopt also prints a diagnostic message to *stderr*.

## Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                       |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | <p>The next option character specified on the command line.</p> <p>A colon is returned if getopt detects a missing argument and the first character of <i>optstring</i> is a colon.</p> <p>A question mark is returned if getopt encounters an option character not in <i>optstring</i> or detects a missing argument and the first character of <i>optstring</i> is not a colon.</p> |
| -1 | When all command-line options are parsed.                                                                                                                                                                                                                                                                                                                                             |

## Example

The following example shows how you might process the arguments for a utility that can take the mutually exclusive options *a* and *b* and the options *f* and *o*, both of which require arguments:

```
#include <unistd.h>

int main (int argc, char *argv[])
{
 int c;
 int bflg, aflg, errflg;
 char *ifile;
 char *ofile;
 extern char *optarg;
 extern int optind, optopt;
 .
 .
 .
 while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
 switch (c) {
 case 'a':
 if (bflg)
 errflg++;
 else
 aflg++;
 break;
 case 'b':
 if (aflg)
 errflg++;
 else {
 bflg++;
 bproc();
 }
 }
 }
}
```

## getopt

```
 break;
 case 'f':
 ifile = optarg;
 break;
 case 'o':
 ofile = optarg;
 break;
 case ':': /* -f or -o without operand */
 fprintf (stderr,
 "Option -%c requires an operand\n" optopt);
 errflg++;
 break;
 case '?':
 fprintf (stderr,
 "Unrecognized option -%c\n" optopt);
 errflg++;
 }
}
if (errflg) {
 fprintf (stderr, "usage: ...");
 exit(2);
}
for (; optind < argc; optind++) {
 if (access(argv[optind], R_OK)) {
 .
 .
 .
 }
}
```

This sample code accepts any of the following as equivalent:

```
cmd -ao arg path path
cmd -a -o arg path path
cmd -o arg -a path path
cmd -a -o arg -- path path
cmd -a -oarg path path
cmd -aoarg path path
```

---

## **getpagesize**

Gets the system page size.

### **Format**

```
#include <unistd.h>

int getpagesize (void);
```

### **Description**

The `getpagesize` function returns the number of bytes in a page. The system page size is useful for specifying arguments to memory management system calls.

The page size is a system page size and is not necessarily the same as the underlying hardware page size.

### **Return Value**

|   |                                                                  |
|---|------------------------------------------------------------------|
| x | Always indicates success. Returns the number of bytes in a page. |
|---|------------------------------------------------------------------|

## getpid

---

## getpid

Returns the process ID of the current process.

### Format

```
#include <unistd.h>

pid_t getpid (void);
```

### Return Value

|   |                                        |
|---|----------------------------------------|
| x | The process ID of the current process. |
|---|----------------------------------------|

---

## **getppid**

Returns the parent process ID of the calling process.

### **Format**

```
#include <unistd.h>

pid_t getppid (void);
```

### **Return Values**

|   |                                                                    |
|---|--------------------------------------------------------------------|
| x | The parent process ID.                                             |
| 0 | Indicates that the calling process does not have a parent process. |

## getpwent

---

## getpwent

Accesses user entry information in the user database.

### Format

```
#include <stdio.h>
#include <pwd.h>
struct passwd *getpwent (void);
```

### Argument

#### **passwd**

The password structure where you write the user attributes.

### Description

The `getpwent` function accesses basic user attributes about a specified user. The function returns the next user entry in a sequential search.

The `passwd` structure is defined in the `<pwd.h>` header file as follows:

|                        |                                                  |
|------------------------|--------------------------------------------------|
| <code>pw_name</code>   | The name of the user.                            |
| <code>pw_passwd</code> | The encrypted password of the user.              |
| <code>pw_uid</code>    | The ID of the user.                              |
| <code>pw_gid</code>    | The group ID of the principle group of the user. |
| <code>pw_pecos</code>  | The personal information about the user.         |
| <code>pw_dir</code>    | The home directory of the user.                  |
| <code>pw_shell</code>  | The initial program for the user.                |

---

#### **Note**

---

All information generated by the `getpwent` function is stored in a per-thread static area and is overwritten on subsequent calls to the function. Password file entries that are too long are ignored.

---

### Return Values

|                   |                                          |
|-------------------|------------------------------------------|
| <code>x</code>    | A pointer to a valid password structure. |
| <code>NULL</code> | Indicates an error.                      |



---

## getpwnam

Accesses user-name information in the user database.

### Format

```
#include <pwd.h>

struct passwd *getpwnam (const char name);
```

### Argument

**name**

The name of the user for which the attributes are to be read.

### Description

The `getpwnam` function returns the first user entry in the database with the `pw_name` member of the `passwd` structure that matches the *name* argument.

The `passwd` structure is defined in the `<pwd.h>` header file as follows:

|                       |                                   |
|-----------------------|-----------------------------------|
| <code>pw_name</code>  | The user's login name.            |
| <code>pw_uid</code>   | The numerical user ID.            |
| <code>pw_gid</code>   | The numerical group ID.           |
| <code>pw_dir</code>   | The home directory of the user.   |
| <code>pw_shell</code> | The initial program for the user. |

---

**Note**

---

All information generated by the `getpwnam` function is stored in a static area and is overwritten on subsequent calls to the function.

---

### Return Values

|                   |                                                                     |
|-------------------|---------------------------------------------------------------------|
| <code>x</code>    | A pointer to a valid password structure.                            |
| <code>NULL</code> | An error occurred. <code>errno</code> is set to indicate the error. |

## getpwuid

---

## getpwuid

Accesses user-ID information in the rights database.

### Format

```
#include <pwd.h>

struct passwd *getpwuid (uid_t uid);
```

### Argument

#### **uid**

The ID of the user for which the attributes are to be read.

### Description

The `getpwuid` function accesses the identifier names of all identifiers in the rights database. It returns the first user entry in the rights database with a `pw_uid` member of the `passwd` structure that matches the *uid* argument.

The `passwd` structure is defined in the `<pwd.h>` header file as follows:

|                       |                                   |
|-----------------------|-----------------------------------|
| <code>pw_name</code>  | The user's login name.            |
| <code>pw_uid</code>   | The numerical user ID.            |
| <code>pw_gid</code>   | The numerical group ID.           |
| <code>pw_dir</code>   | The home directory of the user.   |
| <code>pw_shell</code> | The initial program for the user. |

To check for error situations, applications should set `errno` to zero before calling `getpwuid`. If `errno` is nonzero on return, then an error occurred.

---

#### **Note**

---

All information generated by the `getpwuid` function is stored in a per-thread static area and is overwritten on subsequent calls to the function.

---

### Return Values

|                   |                                                                     |
|-------------------|---------------------------------------------------------------------|
| <code>x</code>    | A pointer to a valid password structure.                            |
| <code>NULL</code> | An error occurred. <code>errno</code> is set to indicate the error. |

---

## gets

Reads a line from the standard input (stdin).

### Format

```
#include <stdio.h>

char *gets (char *str);
```

### Function Variants

The `gets` function has variants named `_gets32` and `_gets64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Argument

#### **str**

A pointer to a character string that is large enough to hold the information fetched from stdin.

### Description

The new-line character ( `\n` ) that ends the line is replaced by the function with an ASCII null character ( `\0` ).

When `stdin` is opened in record mode, `gets` treats the end of a record the same as a new-line character and, therefore, reads up to and including a new-line character or to the end of the record.

### Return Values

x

A pointer to the *str* argument.

NULL

Indicates that an error has occurred or that the end-of-file was encountered before a new-line character was encountered. The contents of *str* are undefined if a read error occurs.

## [w]getstr

---

## [w]getstr

Get a string from the terminal screen, store it in the variable *str*, and echo it on the specified window. The *getstr* function works on the *stdscr* window.

### Format

```
#include <curses.h>

int getstr (char *str);

int wgetstr (WINDOW *win, char *str);
```

### Arguments

**win**

A pointer to the window.

**str**

Must be large enough to hold the character string fetched from the window.

### Description

The *getstr* and *wgetstr* functions refresh the specified window before fetching a string. The new-line terminator is stripped from the fetched string. For more information, see the *scrollok* function in this section.

### Return Values

OK

Indicates success.

ERR

Indicates that the function makes the screen scroll illegally.

## gettimeofday

Gets date and time.

### Format

```
#include <time.h>

int gettimeofday (struct timeval *tp, void *tzp);
```

### Arguments

**tp**

Pointer to a `timeval` structure, defined in the `<time.h>` header file.

**tzp**

A NULL pointer. If this argument is not a NULL pointer, it is ignored.

### Description

The `gettimeofday` function gets the current time (expressed as seconds and microseconds) since 00::00 Coordinated Universal Time, January 1, 1970. The current time is stored in the `timeval` structure pointed to by the `tp` argument.

The `tzp` argument is intended to hold time-zone information set by the kernel. However, because the OpenVMS kernel does not set time-zone information, the `tzp` argument should be NULL. If it is not NULL, it is ignored. This function is supported for compatibility with BSD programs.

If the value of the `SYS$TIMEZONE_DIFFERENTIAL` logical is wrong, the function fails with `errno` set to `EINVAL`.

### Return Values

|    |                                                                     |
|----|---------------------------------------------------------------------|
| 0  | Indicates success.                                                  |
| -1 | An error occurred. <code>errno</code> is set to indicate the error. |

## getuid

---

## getuid

With POSIX IDs disabled, this function is equivalent to `geteuid` and returns the member number (in OpenVMS terms) from the user identification code (UIC).

With POSIX IDs enabled, returns the real user ID.

### Format

```
#include <unistd.h>

uid_t getuid (void);
```

### Description

The `getuid` function can be used with POSIX style identifiers or with UIC-based identifiers.

---

#### Note

---

Currently, POSIX style IDs are supported only by some OpenVMS versions done for specific government agencies, but will be integrated into future OpenVMS releases. OpenVMS Version 7.3-1 does not support POSIX style IDs, but it does support 32-bit identifiers.

---

With POSIX style IDs disabled (the default), the `geteuid` and `getuid` functions are equivalent and return the member number from the current UIC as follows:

- For programs compiled with the `_VMS_V6_SOURCE` feature-test macro or programs that do not include the `<unistd.h>` header file, the `getuid` and `geteuid` functions return the member number of the OpenVMS UIC. For example, if the UIC is [313,31], then the member number, 31, is returned.
- For programs compiled without the `_VMS_V6_SOURCE` feature-test macro that do include the `<unistd.h>` header file, the full UIC is returned. For example, if the UIC is [313, 31] then 20512799 (31 + 313 \* 65536) is returned.

With POSIX style IDs enabled, `geteuid` returns the effective user ID of the calling process, and `getuid` returns the real user ID of the calling process.

See also `getegid` and `getgid` in this section.

### Return Value

|   |                                                                                                                       |
|---|-----------------------------------------------------------------------------------------------------------------------|
| x | The real user ID (POSIX IDs enabled), or the member number from the current UIC or the full UIC (POSIX IDs disabled). |
|---|-----------------------------------------------------------------------------------------------------------------------|

---

## getw

Returns characters from a specified file.

### Format

```
#include <stdio.h>

int getw (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A pointer to the file to be accessed.

### Description

The `getw` function returns the next four characters from the specified input file as an `int`.

### Return Values

|                  |                                                                                                                                                                                                                                                                           |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>   | The next four characters, in an <code>int</code> .                                                                                                                                                                                                                        |
| <code>EOF</code> | Indicates that the end-of-file was encountered during the retrieval of any of the four characters and all four characters were lost. Since <code>EOF</code> is an acceptable integer, use <code>feof</code> and <code>ferror</code> to check the success of the function. |

## getwc

---

## getwc

Reads the next character from a specified file, and converts it to a wide-character code.

### Format

```
#include <wchar.h>

wint_t getwc (FILE *file_ptr);
```

### Argument

**file\_ptr**  
A pointer to the file to be accessed.

### Description

Since `getwc` is implemented as a macro, a file pointer argument with side effects (for example `getwc (*f++)`) might be evaluated incorrectly. In such a case, use the `fgetwc` function instead. See the `fgetwc` function in this section.

### Return Values

|                   |                                                                                                                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>n</code>    | The returned character.                                                                                                                                                                   |
| <code>WEOF</code> | Indicates the end-of-file or an error. If an error occurs, the function sets <code>errno</code> . For a list of the values set by this function, see <code>fgetwc</code> in this section. |



---

## getwchar

Reads a single wide character from the standard input (stdin).

### Format

```
#include <wchar.h>

wint_t getwchar (void);
```

### Description

The `getwchar` function is identical to `fgetwc(stdin)`.

### Return Values

|                   |                                                                                                                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>    | The next character from <code>stdin</code> , converted to <code>wint_t</code> .                                                                                                           |
| <code>WEOF</code> | Indicates the end-of-file or an error. If an error occurs, the function sets <code>errno</code> . For a list of the values set by this function, see <code>fgetwc</code> in this section. |

## getyx

---

## getyx

Puts the (y,x) coordinates of the current cursor position on *win* in the variables *y* and *x*.

### Format

```
#include <curses.h>

getyx (WINDOW *win, int y, int x);
```

### Arguments

**win**  
Must be a pointer to the window.

**y**  
Must be a valid lvalue.

**x**  
Must be a valid lvalue.

---

## gmtime, gmtime\_r

Converts time units to the broken-down UTC time.

### Format

```
#include <time.h>

struct tm *gmtime (const time_t *timer);

struct tm *gmtime_r (const time_t *timer, struct tm *result); (ISO POSIX-1)
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `gmtime_r` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Arguments

#### timer

Points to a variable that specifies a time value in seconds since the Epoch.

#### result

A pointer to a `tm` structure where the result is stored.

The `tm` structure is defined in the `<time.h>` header, and is also shown in Table REF-4 in the description of `localtime`.

### Description

The `gmtime` and `gmtime_r` functions convert the time (in seconds since the Epoch) pointed to by *timer* into a broken-down time, expressed as Coordinated Universal Time (UTC), and store it in a `tm` structure.

The difference between the `gmtime_r` and `gmtime` functions is that the former puts the result into a user-specified `tm` structure where the result is stored. The latter puts the result into thread-specific static memory allocated by the *Compaq C* RTL, and which is overwritten by subsequent calls to `gmtime`; you must make a copy if you want to save it.

On success, `gmtime` returns a pointer to the `tm` structure; `gmtime_r` returns its second argument. On failure, these functions return the NULL pointer.

---

#### Note

---

Generally speaking, UTC-based time functions can affect in-memory time-zone information, which is process-wide data. However, if the system time zone remains the same during the execution of the application (which is the common case) and the cache of timezone files is enabled (which is the default), then the `_r` variant of the time functions `asctime_r`, `ctime_r`, `gmtime_r` and `localtime_r`, is both thread-safe and AST-reentrant.

If, however, the system time zone can change during the execution of the application or the cache of timezone files is not enabled, then both variants of the UTC-based time functions belong to the third class of functions, which are neither thread-safe nor AST-reentrant.

---

## gmtime, gmtime\_r

### Return Values

x

Pointer to a `tm` structure.

NULL

Indicates an error; `errno` is set to one of the following values:

- `EINVAL` – The *timer* argument is NULL.

---

## gsignal

Generates a specified software signal, which invokes the action routine established by a `signal`, `ssignal`, or `sigvec` function.

### Format

```
#include <signal.h>

int gsignal (int sig [, int sigcode]);
```

### Arguments

#### **sig**

The signal to be generated.

#### **sigcode**

An optional signal code. For example, signal SIGFPE—the arithmetic trap signal—has 10 different codes, each representing a different type of arithmetic trap.

The signal codes can be represented by mnemonics or numbers. The arithmetic trap codes are represented by the numbers 1 to 10, but the SIGILL codes are represented by the numbers 0 to 2. The code values are defined in the `<signal.h>` header file. See Tables 4–4 and 4–5 for a list of signal mnemonics, codes, and corresponding OpenVMS exceptions.

### Description

Calling the `gsignal` function has one of the following results:

- If `gsignal` specifies a *sig* argument that is outside the range defined in the `<signal.h>` header file, then `gsignal` returns 0 and sets `errno` to `EINVAL`.
- If `signal`, `ssignal`, or `sigvec` establishes `SIG_DFL` (default action) for the signal, then `gsignal` does not return. The image is exited with the OpenVMS error code corresponding to the signal.
- If `signal`, `ssignal`, or `sigvec` establishes `SIG_IGN` (ignore signal) as the action for the signal, then `gsignal` returns its argument, *sig*.
- `signal`, `ssignal`, or `sigvec` must be used to establish an action routine for the signal. That function is called and its return value is returned by `gsignal`.

See Chapter 4 for more information.

See also `raise`, `signal`, `ssignal`, and `sigvec` in this section.

### Return Values

|            |                                                                                                                                                                        |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0          | Indicates a <i>sig</i> argument that is outside the range defined in the <code>&lt;signal.h&gt;</code> header file; <code>errno</code> is set to <code>EINVAL</code> . |
| <i>sig</i> | Indicates that <code>SIG_IGN</code> (ignore signal) has been established as the action for the signal.                                                                 |

## gsignal

|   |                                                                                                                                                                                                                           |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x | Indicates that <code>signal</code> , <code>ssignal</code> , or <code>sigvec</code> has established an action function for the signal. That function is called, and its return value is returned by <code>gsignal</code> . |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

## hypot

Returns the length of the hypotenuse of a right triangle.

### Format

```
#include <math.h>

double hypot (double x, double y);
float hypotf (float x, float y); (Alpha only)
long double hypotl (long double x, long double y); (Alpha only)
```

### Arguments

**x**  
A real value.

**y**  
A real value.

### Description

The hypot functions return the length of the hypotenuse of a right triangle, where  $x$  and  $y$  represent the perpendicular sides of the triangle. The length is calculated as:

$$\sqrt{x^2 + y^2}$$

On overflow, the return value is undefined, and `errno` is set to `ERANGE`.

### Return Values

|                       |                                                                                   |
|-----------------------|-----------------------------------------------------------------------------------|
| <code>x</code>        | The length of the hypotenuse.                                                     |
| <code>HUGE_VAL</code> | Overflow occurred; <code>errno</code> is set to <code>ERANGE</code> .             |
| <code>0</code>        | Underflow occurred; <code>errno</code> is set to <code>ERANGE</code> .            |
| <code>NaN</code>      | $x$ or $y$ is <code>NaN</code> ; <code>errno</code> is set to <code>EDOM</code> . |

---

## iconv

Converts characters coded in one codeset to characters coded in another codeset.

### Format

```
#include <iconv.h>
```

```
size_t iconv (iconv_t cd, char **inbuf, size_t *inbytesleft, char **outbuf, size_t *outbytesleft);
```

### Arguments

#### **cd**

A conversion descriptor. This is returned by a successful call to `iconv_open`.

#### **inbuf**

A pointer to a variable that points to the first character in the input buffer.

#### **inbytesleft**

Initially, this argument is a pointer to a variable that indicates the number of bytes to the end of the input buffer (*inbuf*). When the conversion is completed, the variable indicates the number of bytes in *inbuf* not converted.

#### **outbuf**

A pointer to a variable that points to the first available byte in the output buffer. The output buffer contains the converted characters.

#### **outbytesleft**

Initially, this argument is a pointer to a variable that indicates the number of bytes to the end of the output buffer (*outbuf*). When the conversion is completed, the variable indicates the number of bytes left in *outbuf*.

### Description

The `iconv` function converts characters in the buffer pointed to by *inbuf* to characters in another code set. The resulting characters are stored in the buffer pointed to by *outbuf*. The conversion type is specified by the conversion descriptor *cd*. This descriptor is returned from a successful call to `iconv_open`.

If an invalid character is found in the input buffer, the conversion stops after the last successful conversion. The variable pointed to by *inbytesleft* is updated to reflect the number of bytes in the input buffer that are not converted. The variable pointed to by *outbytesleft* is updated to reflect the number of bytes remaining in the output buffer.

### Return Values

x

Number of nonidentical conversions performed. Indicates successful conversion. In most cases, 0 is returned.



(size\_t) - 1

Indicates an error condition. The function sets `errno` to one of the following:

- `EBADF` – The *cd* argument is not a valid conversion descriptor.
- `EILSEQ` – The conversion stops when an invalid character detected.
- `E2BIG` – The conversion stops because of insufficient space in the output buffer.
- `EINVAL` – The conversion stops because of an incomplete character at the end of the input buffer.

## iconv\_close

---

## iconv\_close

Deallocates a specified conversion descriptor and the resources allocated to the descriptor.

### Format

```
#include <iconv.h>

int iconv_close (iconv_t cd);
```

### Argument

**cd**

The conversion descriptor to be deallocated. A conversion descriptor is returned by a successful call to `iconv_open`.

### Return Values

- |    |                                                                                                                                                                                                                                                                                                                                             |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates that the conversion descriptor was successfully deallocated.                                                                                                                                                                                                                                                                      |
| -1 | Indicates an error occurred. The function sets <code>errno</code> to one of the following: <ul style="list-style-type: none"><li>• <b>EBADF</b> – The <i>cd</i> argument is not a valid conversion descriptor.</li><li>• <b>EVMSEERR</b> – Nontranslatable VMS error occur. <code>vaxc\$errno</code> contains the VMS error code.</li></ul> |

---

## iconv\_open

Allocates a conversion descriptor for a specified codeset conversion.

### Format

```
#include <iconv.h>

iconv_t iconv_open (const char *tocode, const char *fromcode);
```

### Arguments

#### **to**code

The name of the codeset to which characters are converted.

#### **from**code

The name of the source codeset. See Chapter 10 for information on obtaining a list of currently available codesets or for details on adding new codesets.

### Return Values

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x            | A conversion descriptor. Indicates the call was successful. This descriptor is used in subsequent calls to <code>iconv</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| (iconv_t) -1 | Indicates an error occurred. The function sets <code>errno</code> to one of the following: <ul style="list-style-type: none"> <li>• <b>EMFILE</b> – The process does not have enough I/O channels to open a file.</li> <li>• <b>ENOMEM</b> – Insufficient space is available.</li> <li>• <b>EINVAL</b> – The conversion specified by <i>fromcode</i> and <i>to</i>code is not supported.</li> <li>• <b>EVMSError</b> – Nontranslatable VMS error occur. <code>vaxc\$errno</code> contains the VMS error code. A value of <code>SS\$_BADCHKSUM</code> in <code>vaxc\$errno</code> indicates that a conversion table file was found, but its contents is corrupted. A value of <code>SS\$_IDMISMATCH</code> in <code>vaxc\$errno</code> indicates that the conversion table file version does not match the version of the C Run-Time Library.</li> </ul> |

### Example

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

int main()
{
 /* Declare variables to be used */
 /* */
```

## iconv\_open

```
char fromcodeset[30];
char tocodeset[30];
int iconv_opened;
iconv_t iconv_struct; /* Iconv descriptor */
/* Initialize variables */
sprintf(fromcodeset, "DECHANYU");
sprintf(tocodeset, "EUCTW");
iconv_opened = FALSE;

/* Attempt to create a conversion descriptor for the */
/* codesets specified. If the return value from */
/* iconv_open is -1 then an error has occurred. */
/* Check the value of errno. */
if ((iconv_struct = iconv_open(tocodeset, fromcodeset))
 == (iconv_t) - 1) {
 /* Check the value of errno */

 switch (errno) {
 case EMFILE:
 case ENFILE:
 printf("Too many iconv conversion files open\n");
 break;

 case ENOMEM:
 printf("Not enough memory\n");
 break;

 case EINVAL:
 printf("Unsupported conversion\n");
 break;

 default:
 printf("Unexpected error from iconv_open\n");
 break;
 }
}
else
 /* Successfully allocated a conversion descriptor */
 iconv_opened = TRUE;

/* Was a conversion descriptor allocated */
if (iconv_opened) {
 /* Attempt to deallocate the conversion descriptor. */
 /* If iconv_close returns -1 then an error has */
 /* occurred. */
 if (iconv_close(iconv_struct) == -1) {
 /* An error occurred. Check the value of errno */
 switch (errno) {
 case EBADF:
 printf("Conversion descriptor is invalid\n");
 break;
 default:
 printf("Unexpected error from iconv_close\n");
 break;
 }
 }
}
return (EXIT_FAILURE);
}
```

---

**[w]inch**

Return the character at the current cursor position on the specified window without making changes to the window. The `inch` function acts on the `stdscr` window.

**Format**

```
#include <curses.h>

char inch();

char winch (WINDOW *win);
```

**Argument**

**win**  
A pointer to the window.

**Return Values**

|     |                           |
|-----|---------------------------|
| x   | The returned character.   |
| ERR | Indicates an input error. |

---

## index

Search for a character in a string.

### Format

```
#include <strings.h>

char *index (const char *s, int c);
```

### Function Variants

The `index` function has variants named `_index32` and `_index64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**s**  
The string to search.

**c**  
The character to search for.

### Description

The `index` function is identical to the `strchr` function, and is provided for compatibility with some UNIX implementations.

---

## initscr

Initializes the terminal-type data and all screen functions. You must call `initscr` before using any of the curses functions.

### Format

```
#include <curses.h>

void initscr (void);
```

### Description

The OpenVMS Curses version of the `initscr` function clears the screen before doing the initialization. The BSD-based Curses version does not.

---

## initstate

Initializes random-number generators.

### Format

```
#include <stdlib.h>

char *initstate (unsigned int seed, char *state, int size);
```

### Arguments

**seed**

An initial seed value.

**state**

Pointer to an array of state information.

**size**

The size of the state information array.

### Description

The `initstate` function initializes random-number generators. It lets you initialize, for future use, a state array passed as an argument. The size, in bytes, of the state array is used by the `initstate` function to decide how sophisticated a random-number generator to use; the larger the state array, the more random the numbers.

Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Amounts less than 8 bytes generate an error, while other amounts are rounded down to the nearest known value.

The *seed* argument specifies a starting point for the random-number sequence and provides for restarting at the same point. The `initstate` function returns a pointer to the previous state information array.

Once you initialize a state, the `setstate` function allows rapid switching between states. The array defined by the *state* argument is used for further random-number generation until the `initstate` function is called or the `setstate` function is called again. The `setstate` function returns a pointer to the previous state array.

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

- Use the `initstate` function with the desired *seed* argument, *state* array, and *size* of the array.
- Use the `setstate` function with the desired state, followed by the `random` function with the desired *seed*. The advantage of using both functions is that you do not have to save the state array size once you initialize it.

See also `setstate`, `random`, and `rand` in this section.



**Return Values**

|   |                                                                                                                   |
|---|-------------------------------------------------------------------------------------------------------------------|
| x | A pointer to the previous state array information.                                                                |
| 0 | Indicates an error. Call made with less than 8 bytes of state information. Further specified in the global errno. |

## [w]insch

---

## [w]insch

Insert a character at the current cursor position in the specified window. The `insch` function acts on the `stdscr` window.

### Format

```
#include <curses.h>

int insch (char ch);

int winsch (WINDOW *win, char ch);
```

### Arguments

**win**

A pointer to the window.

**ch**

The character to be inserted.

### Description

After the character is inserted, each character on the line shifts to the right, and the last character in the line is deleted. For more information, see the `scrollok` function in this section.

### Return Values

OK

Indicates success.

ERR

Indicates that the function makes the screen scroll illegally.

---

**[w]insertln**

Insert a line above the line containing the current cursor position. The `insertln` function acts on the `stdscr` window.

**Format**

```
#include <curses.h>

int insertln();

int wininsertln (WINDOW *win);
```

**Argument**

**win**  
A pointer to the window.

**Description**

The current line and every line below it shifts down, and the bottom line disappears. The inserted line is blank and the current (y,x) coordinates remain the same. For more information, see the `scrollok` function in this section.

**Return Values**

|     |                                                                |
|-----|----------------------------------------------------------------|
| OK  | Indicates success.                                             |
| ERR | Indicates that the function makes the screen scroll illegally. |

## [w]insstr

---

## [w]insstr

Insert a string at the current cursor position in the specified window. The `insstr` function acts on the `stdscr` window.

### Format

```
#include <curses.h>

int insstr (char *str);

int winsstr (WINDOW *win, char *str);
```

### Arguments

**win**

A pointer to the window.

**str**

A pointer to the string to be inserted.

### Description

Each character after the string shifts to the right, and the last character disappears. These functions are specific to *Compaq C* for *OpenVMS* Systems and are not portable.

### Return Values

OK

Indicates success.

ERR

Indicates that the function makes the screen scroll illegally. For more information, see the `scrollok` function in this section.

---

## isalnum

Indicates if a character is classed either as alphabetic or as a digit in the program's current locale.

### Format

```
#include <ctype.h>

int isalnum (int character);
```

### Argument

#### **character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                      |
|---------|----------------------|
| nonzero | If alphanumeric.     |
| 0       | If not alphanumeric. |

## isalpha

---

## isalpha

Indicates if a character is classed as an alphabetic character in the program's current locale.

### Format

```
#include <ctype.h>

int isalpha (int character);
```

### Argument

#### **character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                    |
|---------|--------------------|
| nonzero | If alphabetic.     |
| 0       | If not alphabetic. |

---

## isapipe

Indicates if a specified file descriptor is associated with a pipe.

### Format

```
#include <unistd.h>

int isapipe (int file_desc);
```

### Argument

**file\_desc**  
A file descriptor.

### Description

For more information about pipes, see Chapter 5.

### Return Values

|    |                                                                                               |
|----|-----------------------------------------------------------------------------------------------|
| 1  | Indicates an association with a pipe.                                                         |
| 0  | Indicates no association with a pipe.                                                         |
| -1 | Indicates an error (for example, if the file descriptor is not associated with an open file). |

## isascii

---

## isascii

Indicates if a character is an ASCII character.

### Format

```
#include <ctype.h>

int isascii (int character);
```

### Argument

**character**  
An object of type char.

### Return Values

|         |               |
|---------|---------------|
| nonzero | If ASCII.     |
| 0       | If not ASCII. |



---

## isatty

Indicates if a specified file descriptor is associated with a terminal.

### Format

```
#include <unistd.h>

int isatty (int file_desc);
```

### Argument

**file\_desc**  
A file descriptor.

### Return Values

|    |                                                                                               |
|----|-----------------------------------------------------------------------------------------------|
| 1  | If the file descriptor is associated with a terminal.                                         |
| 0  | If the file descriptor is not associated with a terminal.                                     |
| −1 | Indicates an error (for example, if the file descriptor is not associated with an open file). |

## isctrl

---

## isctrl

Indicates if a character is classed as a control character in the program's current locale.

### Format

```
#include <ctype.h>

int isctrl (int character);
```

### Argument

#### **character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                             |
|---------|-----------------------------|
| nonzero | If a control character.     |
| 0       | If not a control character. |

---

## isdigit

Indicates if a character is classed as a digit in the program's current locale.

### Format

```
#include <ctype.h>

int isdigit (int character);
```

### Argument

#### **character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                         |
|---------|-------------------------|
| nonzero | If a decimal digit.     |
| 0       | If not a decimal digit. |

## isgraph

---

## isgraph

Indicates if a character is classed as a graphic character in the program's current locale.

### Format

```
#include <ctype.h>

int isgraph (int character);
```

### Argument

#### **character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                             |
|---------|-----------------------------|
| nonzero | If a graphic character.     |
| 0       | If not a graphic character. |

---

## islower

Indicates if a character is classed as a lowercase character in the program's current locale.

### Format

```
#include <ctype.h>

int islower (int character);
```

### Argument

**character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                                          |
|---------|------------------------------------------|
| nonzero | If a lowercase alphabetic character.     |
| 0       | If not a lowercase alphabetic character. |

## **isnan** *(Alpha only)*

---

## **isnan** *(Alpha only)*

Tests for a NaN. Returns 1 if the argument is NaN; 0 if not.

### **Format**

```
#include <math.h>

int isnan (double x);
int isnanf (float x);
int isnanl (long double x);
```

### **Argument**

**x**  
A real value.

### **Description**

The `isnan` functions return the integer value 1 (TRUE) if *x* is NaN (the IEEE floating point reserved not-a-number value); otherwise, they return the value 0 (FALSE).

---

## isprint

Indicates if a character is classed as a printing character in the program's current locale.

### Format

```
#include <ctype.h>

int isprint (int character);
```

### Argument

**character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                              |
|---------|------------------------------|
| nonzero | If a printing character.     |
| 0       | If not a printing character. |

## ispunct

---

## ispunct

Indicates if a character is classed as a punctuation character in the program's current locale.

### Format

```
#include <ctype.h>

int ispunct (int character);
```

### Argument

#### **character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                                 |
|---------|---------------------------------|
| nonzero | If a punctuation character.     |
| 0       | If not a punctuation character. |



---

## isspace

Indicates if a character is classed as white space in the program's current locale; that is, if it is an ASCII space, tab (horizontal or vertical), carriage-return, form-feed, or new-line character.

### Format

```
#include <ctype.h>

int isspace (int character);
```

### Argument

**character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                                 |
|---------|---------------------------------|
| nonzero | If a white-space character.     |
| 0       | If not a white-space character. |

## isupper

---

## isupper

Indicates if a character is classed as an uppercase character in the program's current locale.

### Format

```
#include <ctype.h>

int isupper (int character);
```

### Argument

#### **character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                                           |
|---------|-------------------------------------------|
| nonzero | If an uppercase alphabetic character.     |
| 0       | If not an uppercase alphabetic character. |

---

## iswalnum

Indicates if a wide character is classed either as alphabetic or as a digit in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswalnum (wint_t wc);
```

### Argument

**wc**

An object of type `wint_t`. The value of *character* must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                      |
|---------|----------------------|
| nonzero | If alphanumeric.     |
| 0       | If not alphanumeric. |

## iswalpha

---

## iswalpha

Indicates if a wide character is classed as an alphabetic character in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)

int iswalpha (wint_t wc);
```

### Argument

**wc**

An object of type `wint_t`. The value of `wc` must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                    |
|---------|--------------------|
| nonzero | If alphabetic.     |
| 0       | If not alphabetic. |

---

## iswcntrl

Indicates if a wide character is classed as a control character in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswcntrl (wint_t wc);
```

### Argument

**wc**

An object of type `wint_t`. The value of `wc` must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

nonzero

If a control character.

0

If not a control character.

---

## iswctype

Indicates if a wide character has a specified property.

### Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)

int iswctype (wint_t wc, wctype_t wc_prop);
```

### Arguments

#### **wc**

An object of type `wint_t`. The value of *wc* must be representable as a valid wide-character code in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

#### **wc\_prop**

A valid property name in the current locale. This is set up by calling the `wctype` function.

### Description

The `iswctype` function tests whether *wc* has the character-class property *wc\_prop*. Set *wc\_prop* by calling the `wctype` function.

See also `wctype` in this section.

### Return Values

|         |                                                              |
|---------|--------------------------------------------------------------|
| nonzero | If the character has the property <i>wc_prop</i> .           |
| 0       | If the character does not have the property <i>wc_prop</i> . |

### Example

```
#include <locale.h>
#include <wchar.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

/* This test will set up the "upper" character class using */
/* wctype() and then verify whether the characters 'a' and 'A' */
/* are members of this class */
#include <stdlib.h>

main()
{
 wchar_t w_char1,
 w_char2;
 wctype_t ret_val;

 char *char1 = "a";
 char *char2 = "A";

 ret_val = wctype("upper");
```

```

/* Convert char1 to wide-character format - w_char1 */
if (mbtowc(&w_char1, char1, 1) == -1) {
 perror("mbtowc");
 exit(EXIT_FAILURE);
}
if (iswctype((wint_t) w_char1, ret_val))
 printf("[%C] is a member of the character class upper\n",
 w_char1);
else
 printf("[%C] is not a member of the character class upper\n",
 w_char1);
/* Convert char2 to wide-character format - w_char2 */
if (mbtowc(&w_char2, char2, 1) == -1) {
 perror("mbtowc");
 exit(EXIT_FAILURE);
}
if (iswctype((wint_t) w_char2, ret_val))
 printf("[%C] is a member of the character class upper\n",
 w_char2);
else
 printf("[%C] is not a member of the character class upper\n",
 w_char2);
}

```

Running the example program produces the following result:

```

[a] is not a member of the character class upper
[A] is a member of the character class upper

```

## iswdigit

---

## iswdigit

Indicates if a wide character is classed as a digit in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswdigit (wint_t wc);
```

### Argument

#### ***wc***

An object of type `wint_t`. The value of *wc* must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                         |
|---------|-------------------------|
| nonzero | If a decimal digit.     |
| 0       | If not a decimal digit. |



---

## iswgraph

Indicates if a wide character is classed as a graphic character in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswgraph (wint_t wc);
```

### Argument

***wc***

An object of type `wint_t`. The value of *wc* must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

nonzero

If a graphic character.

0

If not a graphic character.

## iswlower

---

## iswlower

Indicates if a wide character is classed as a lowercase character in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswlower (wint_t wc);
```

### Argument

**wc**

An object of type `wint_t`. The value of `wc` must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                               |
|---------|-------------------------------|
| nonzero | If a lowercase character.     |
| 0       | If not a lowercase character. |

---

## iswprint

Indicates if a wide character is classed as a printing character in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswprint (wint_t wc);
```

### Argument

**wc**

An object of type `wint_t`. The value of `wc` must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                              |
|---------|------------------------------|
| nonzero | If a printing character.     |
| 0       | If not a printing character. |

## iswpunct

---

## iswpunct

Indicates if a wide character is classed as a punctuation character in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswpunct (wint_t wc);
```

### Argument

#### ***wc***

An object of type `wint_t`. The value of *wc* must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                                 |
|---------|---------------------------------|
| nonzero | If a punctuation character.     |
| 0       | If not a punctuation character. |

---

## iswspace

Indicates if a wide character is classed as a space character in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswspace (wint_t wc);
```

### Argument

**wc**

An object of type `wint_t`. The value of `wc` must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

nonzero

If a white-space character.

0

If not a white-space character.

## iswupper

---

## iswupper

Indicates if a wide character is classed as an uppercase character in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswupper (wint_t wc);
```

### Argument

#### **wc**

An object of type `wint_t`. The value of `wc` must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                                |
|---------|--------------------------------|
| nonzero | If an uppercase character.     |
| 0       | If not an uppercase character. |

---

## iswxdigit

Indicates if a wide character is a hexadecimal digit (0 to 9, A to F, or a to f) in the program's current locale.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int iswxdigit (wint_t wc);
```

### Argument

***wc***

An object of type `wint_t`. The value of *wc* must be representable as a `wchar_t` in the current locale, or must equal the value of the macro `WEOF`. If it has any other value, the behavior is undefined.

### Return Values

|         |                             |
|---------|-----------------------------|
| nonzero | If a hexadecimal digit.     |
| 0       | If not a hexadecimal digit. |

## isxdigit

---

### isxdigit

Indicates if a character is a hexadecimal digit (0 to 9, A to F, or a to f) in the program's current locale.

#### Format

```
#include <ctype.h>

int isxdigit (int character);
```

#### Argument

**character**

An object of type `int`. The value of *character* must be representable as an unsigned `char` in the current locale, or must equal the value of the macro `EOF`. If it has any other value, the behavior is undefined.

#### Return Values

|         |                             |
|---------|-----------------------------|
| nonzero | If a hexadecimal digit.     |
| 0       | If not a hexadecimal digit. |



---

**j0, j1, jn** *(Alpha only)*

Compute Bessel functions of the first kind.

**Format**

```
#include <math.h>
double j0 (double x);
float j0f (float x);
long double j0l (long double x);
double j1 (double x);
float j1f (float x);
long double j1l (long double x);
double jn (int n, double x);
float jnf (int n, float x);
long double jnl (int n, long double x);
```

**Arguments**

**x**  
A real value.

**n**  
An integer.

**Description**

The `j0` functions return the value of the Bessel function of the first kind of order 0.

The `j1` functions return the value of the Bessel function of the first kind of order 1.

The `jn` functions return the value of the Bessel function of the first kind of order  $n$ .

The `j1` and `jn` functions can result in an underflow as  $x$  gets small. The largest value of  $x$  for which this occurs is a function of  $n$ .

**Return Values**

|                |                                                                                                                       |
|----------------|-----------------------------------------------------------------------------------------------------------------------|
| <code>x</code> | The relevant Bessel value of $x$ of the first kind.                                                                   |
| 0              | The value of the $x$ argument is too large, or underflow occurred; <code>errno</code> is set to <code>ERANGE</code> . |
| NaN            | $x$ is NaN; <code>errno</code> is set to <code>EDOM</code> .                                                          |

---

## jrand48

Generate uniformly distributed pseudorandom-number sequences. Returns 48-bit signed, long integers.

### Format

```
#include <stdlib.h>

long int jrand48 (unsigned short int xsubi[3]);
```

### Argument

#### **xsubi**

An array of three short int that form a 48-bit integer when concatenated together.

### Description

The jrand48 function generates pseudorandom numbers using the linear congruential algorithm and 48-bit integer arithmetic.

The function returns signed long integers uniformly distributed over the range of y values, such that  $-2^{31} \leq y < 2^{31}$ .

The function works by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

The argument  $m$  equals  $2^{48}$ , so 48-bit integer arithmetic is performed. Unless you invoke the lcong48 function, the multiplier value  $a$  and the addend value  $c$  are:

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

The jrand48 function requires that the calling program pass an array as the *xsubi* argument, which for the first call must be initialized to the initial value of the pseudorandom-number sequence. Unlike the drand48 function, it is not necessary to call an initialization function prior to the first call.

By using different arguments, jrand48 allows separate modules of a large program to generate several independent sequences of pseudorandom numbers. For example, the sequence of numbers that one module generates does not depend upon how many times the function is called by other modules.

### Return Value

|   |                                                                                        |
|---|----------------------------------------------------------------------------------------|
| n | Signed, long integers uniformly distributed over the range $-2^{31} \leq y < 2^{31}$ . |
|---|----------------------------------------------------------------------------------------|

---

## kill

Sends a signal to the process specified by a process ID.

### Format

```
#include <signal.h>

int kill (int pid, int sig);
```

### Arguments

**pid**  
The process ID.

**sig**  
The signal code.

### Description

The kill function is restricted to C and C++ programs that include the main function.

The kill function sends a signal to a process, as if the process had called raise. If the signal is not trapped or ignored by the target program, the program exits.

OpenVMS VAX and Alpha implement different rules about what process you are allowed to send signals to. A program always has privileges to send a signal to a child started with vfork/exec. For other processes, the results are determined by the OpenVMS security model for your system.

Because of an OpenVMS restriction, the kill function cannot deliver a signal to a target process that runs an image installed with privileges.

Unless you have system privileges, the sending and receiving processes must have the same user identification code (UIC).

On OpenVMS systems before Version 7.0, kill treats a signal value of 0 as if SIGKILL were specified.

For OpenVMS Version 7.0 and higher systems, if you include <stdlib.h> and compile with the \_POSIX\_EXIT feature-test macro set, then:

- If the signal value is 0, kill validates the process ID but does not send any signals.
- If the process ID is not valid, kill returns -1 and sets errno to ESRCH.

### Return Values

|    |                                                                                                                                          |
|----|------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates that kill was successfully queued.                                                                                             |
| -1 | Indicates errors. The receiving process may have a different UIC and you are not a system user, or the receiving process does not exist. |

## labs

---

## labs

Returns the absolute value of an integer as a long int.

### Format

```
#include <stdlib.h>

long int labs (long int j);
```

### Argument

**j**  
A value of type long int.

---

## lcong48

Initializes a 48-bit uniformly distributed pseudorandom-number sequence.

### Format

```
#include <stdlib.h>

void lcong48 (unsigned short int param[7]);
```

### Argument

#### **param**

An array that in turn specifies the initial  $X_i$ , the multiplier value  $a$ , and the addend value  $c$ .

### Description

The `lcong48` function generates pseudorandom numbers using the linear congruential algorithm and 48-bit integer arithmetic.

You can use `lcong48` to initialize the random number generator before you call any of the following functions:

```
drand48
lrand48
mrand48
```

The `lcong48` function specifies the initial  $X_i$  value, the multiplier value  $a$ , and the addend value  $c$ . The *param* array elements specify the following:

|                    |                         |
|--------------------|-------------------------|
| <i>param</i> [0-2] | $X_i$                   |
| <i>param</i> [3-5] | Multiplier $a$ value    |
| <i>param</i> [6]   | 16-bit addend $c$ value |

After `lcong48` has been called, a subsequent call to either `srand48` or `seed48` restores the standard  $a$  and  $c$  as specified previously.

The `lcong48` function does not return a value.

See also `drand48`, `lrand48`, `mrand48`, `srand48`, and `seed48` in this section.

## ldexp

---

## ldexp

Returns its first argument multiplied by 2 raised to the power of its second argument; that is,  $x(2^n)$ .

### Format

```
#include <math.h>

double ldexp (double x, int n);
float ldexp (float x, int n); (Alpha only)
long double ldexp (long double x, int n); (Alpha only)
```

### Arguments

**x**  
A base value of type double, float, or long double that is to be multiplied by  $2^n$ .

**n**  
The integer exponent value to which 2 is raised.

### Return Values

|          |                                                                                |
|----------|--------------------------------------------------------------------------------|
| $x(2^n)$ | The first argument multiplied by 2 raised to the power of the second argument. |
| 0        | Underflow occurred; errno is set to ERANGE.                                    |
| HUGE_VAL | Overflow occurred; errno is set to ERANGE.                                     |
| NaN      | $x$ is NaN; errno is set to EDOM.                                              |

---

## ldiv

Returns the quotient and the remainder after the division of its arguments.

### Format

```
#include <stdlib.h>

ldiv_t ldiv (long int numer, long int denom);
```

### Arguments

**numer**

A numerator of type long int.

**denom**

A denominator of type long int.

### Description

The type `ldiv_t` is defined in the `<stdlib.h>` header file as follows:

```
typedef struct
{
 long quot, rem;
} ldiv_t;
```

See also `div` in this section.

---

## leaveok

Signals Curses to leave the cursor at the current coordinates after an update to the window.

### Format

```
#include <curses.h>

leaveok (WINDOW *win, bool boolf);
```

### Arguments

#### **win**

A pointer to the window.

#### **boolf**

A Boolean TRUE or FALSE value. If *boolf* is TRUE, the cursor remains in place after the last update and the coordinate setting on *win* changes accordingly. If *boolf* is FALSE, the cursor moves to the currently specified (y,x) coordinates of *win*.

### Description

The `leaveok` function defaults to moving the cursor to the current coordinates of *win*. The `bool` type is defined in the `<curses.h>` header file as follows:

```
#define bool int
```



---

**lgamma** *(Alpha only)*

Computes the logarithm of the gamma function.

**Format**

```
#include <math.h>

double lgamma (double x);
float lgammaf (float x);
long double lgammal (long double x);
```

**Argument**

**x**  
A real number.  $x$  cannot be 0, a negative integer, or Infinity.

**Description**

The lgamma functions return the logarithm of the absolute value of gamma of  $x$ , or  $\ln(|\Gamma(x)|)$ , where  $\Gamma$  is the gamma function.

The sign of gamma of  $x$  is returned in the external integer variable `signgam`. The  $x$  argument cannot be 0, a negative integer, or Infinity.

**Return Values**

|                        |                                                                                            |
|------------------------|--------------------------------------------------------------------------------------------|
| <code>x</code>         | The logarithmic gamma of the $x$ argument.                                                 |
| <code>-HUGE_VAL</code> | The $x$ argument is a negative integer; <code>errno</code> is set to <code>ERANGE</code> . |
| <code>NaN</code>       | The $x$ argument is <code>NaN</code> ; <code>errno</code> is set to <code>EDOM</code> .    |
| <code>0</code>         | Underflow occurred; <code>errno</code> is set to <code>ERANGE</code> .                     |
| <code>HUGE_VAL</code>  | Overflow occurred; <code>errno</code> is set to <code>ERANGE</code> .                      |

## link

---

## link

Creates a new link (directory entry) for an existing file. This function is supported only on volumes that have hard link counts enabled.

### Format

```
#include <unistd.h>

link (const char *path1, const char *path2);
```

### Arguments

**path1**

Pointer to a pathname naming an existing file.

**path2**

Pointer to a pathname naming the new directory entry to be created.

### Description

The link function atomically creates a new link for the existing file, and the link count of the file is incremented by one.

The link function can be used on directory files.

If link fails, no link is created and the link count of the file remains unchanged.

### Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Successful completion.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| -1 | Indicates an error. The function sets errno to one of the following values: <ul style="list-style-type: none"><li>• EEXIST – The link named by <i>path2</i> exists.</li><li>• EFTYPE – Wildcards appear in either <i>path1</i> or <i>path2</i>.</li><li>• EINVAL – One or both arguments specify a syntactically invalid pathname.</li><li>• ENAMETOOLONG – The length of <i>path1</i> or <i>path2</i> exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX}.</li><li>• EXDEV – The link named by <i>path2</i> and the file named by <i>path1</i> are on different devices.</li></ul> |

---

## localeconv

Sets the members of a structure of type `struct lconv` with values appropriate for formatting numeric quantities according to the rules of the current locale.

### Format

```
#include <locale.h>

struct lconv *localeconv (void);
```

### Description

The `localeconv` function returns a pointer to the `lconv` structure defined in the `<locale.h>` header file. This structure should not be modified by the program. It is overwritten by calls to `localeconv`, or by calls to the `setlocale` function that change the `LC_NUMERIC`, `LC_MONETARY`, or `LC_ALL` categories.

The members of the structure are:

| Member                               | Description                                                                                                                                                             |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>char *decimal_point</code>     | The radix character.                                                                                                                                                    |
| <code>char *thousands_sep</code>     | The character used to separate groups of digits.                                                                                                                        |
| <code>char *grouping</code>          | The string that defines how digits are grouped in nonmonetary values.                                                                                                   |
| <code>char *int_curr_symbol</code>   | The international currency symbol.                                                                                                                                      |
| <code>char *currency_symbol</code>   | The local currency symbol.                                                                                                                                              |
| <code>char *mon_decimal_point</code> | The radix character used to format monetary values.                                                                                                                     |
| <code>char *mon_thousands_sep</code> | The character used to separate groups of digits in monetary values.                                                                                                     |
| <code>char *mon_grouping</code>      | The string that defines how digits are grouped in a monetary value.                                                                                                     |
| <code>char *positive_sign</code>     | The string used to indicate a nonnegative monetary value.                                                                                                               |
| <code>char *negative_sign</code>     | The string used to indicate a negative monetary value.                                                                                                                  |
| <code>char int_frac_digits</code>    | The number of digits displayed after the radix character in a monetary value formatted with the international currency symbol.                                          |
| <code>char frac_digits</code>        | The number of digits displayed after the radix character in a monetary value.                                                                                           |
| <code>char p_cs_precedes</code>      | For positive monetary values, this is set to 1 if the local or international currency symbol precedes the number, and it is set to 0 if the symbol succeeds the number. |

| Member              | Description                                                                                                                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| char p_sep_by_space | For positive monetary values, this is set to 0 if there is no space between the currency symbol and the number. It is set to 1 if there is a space, and it is set to 2 if there is a space between the symbol and the sign string. |
| char n_cs_precedes  | For negative monetary values, this is set to 1 if the local or international currency symbol precedes the number, and it is set to 0 if the symbol succeeds the number.                                                            |
| char n_sep_by_space | For negative monetary values, this is set to 0 if there is no space between the currency symbol and the number. It is set to 1 if there is a space, and it is set to 2 if there is a space between the symbol and the sign string. |
| char p_sign_posn    | An integer used to indicate where the positive_sign string should be placed for a nonnegative monetary quantity.                                                                                                                   |
| char n_sign_posn    | An integer used to indicate where the negative_sign string should be placed for a negative monetary quantity.                                                                                                                      |

Members of the structure of type `char*` are pointers to strings, any of which (except `decimal_point`) can point to "", indicating that the associated value is not available in the current locale or is zero length. Members of the structure of type `char` are positive numbers, any of which can be `CHAR_MAX`, indicating that the associated value is not available in the current locale. `CHAR_MAX` is defined in the `<limits.h>` header file.

Be aware that the value of the `CHAR_MAX` macro in the `<limits.h>` header depends on whether the program is compiled with the `/UNSIGNED_CHAR` qualifier:

- Use the `CHAR_MAX` macro as an indicator of a non-available value in the current locale only if the program is compiled *without* `/UNSIGNED_CHAR` (`/NOUNSIGNED_CHAR` is the default).
- If the program is compiled *with* `/UNSIGNED_CHAR`, use the `SCHAR_MAX` macro instead of the `CHAR_MAX` macro.

In `/NOUNSIGNED_CHAR` mode, the values of `CHAR_MAX` and `SCHAR_MAX` are the same; therefore, comparison with `SCHAR_MAX` gives correct results regardless of the `/[NO]UNSIGNED_CHAR` mode used.

The members `grouping` and `mon_grouping` point to a string that defines the size of each group of digits when formatting a number. Each group size is separated by a semicolon (;). For example, if `grouping` points to the string `5;3` and the `thousands_sep` character is a comma (,), the number `123450000` would be formatted as `1,234,50000`.

The elements of `grouping` and `mon_grouping` are interpreted as follows:

| Value    | Interpretation                                                                                                                                                                      |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CHAR_MAX | No further grouping is performed.                                                                                                                                                   |
| 0        | The previous element is to be used repeatedly for the remainder of the digits.                                                                                                      |
| other    | The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group. |

The values of p\_sign\_posn and n\_sign\_posn are interpreted as follows:

| Value | Interpretation                                                       |
|-------|----------------------------------------------------------------------|
| 0     | Parentheses surround the number and currency symbol.                 |
| 1     | The sign string precedes the number and currency symbol.             |
| 2     | The sign string succeeds the number and currency symbol.             |
| 3     | The sign string immediately precedes the number and currency symbol. |
| 4     | The sign string immediately succeeds the number and currency symbol. |

## Return Value

|   |                                 |
|---|---------------------------------|
| x | Pointer to the lconv structure. |
|---|---------------------------------|

### Example

```
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <locale.h>
#include <string.h>

/* The following test program will set up the British English
/* locale, and then extract the International Currency symbol
/* and the International Fractional Digits fields for this
/* locale and print them. */

int main()
{
 /* Declare variables */
 char *return_val;
 struct lconv *lconv_ptr;

 /* Load a locale */
 return_val = (char *) setlocale(LC_ALL, "en_GB.iso8859-1");
 /* Did the locale load successfully?
 if (return_val == NULL) {
 /* It failed to load the locale
 printf("ERROR : The locale is unknown");
 exit(EXIT_FAILURE);
 }

 /* Get the lconv structure from the locale
 lconv_ptr = (struct lconv *) localeconv();
```

## localeconv

```
/* Compare the international currency symbol string with an */
/* empty string. If they are equal, then the international */
/* currency symbol is not defined in the locale. */
if (strcmp(lconv_ptr->int_curr_symbol, "")) {
 printf("International Currency Symbol = %s\n",
 lconv_ptr->int_curr_symbol);
}
else {
 printf("International Currency Symbol =");
 printf("[Not available in this locale]\n");
}

/* Compare International Fractional Digits with CHAR_MAX. */
/* If they are equal, then International Fractional Digits */
/* are not defined in this locale. */
if ((unsigned char) (lconv_ptr->int_frac_digits) != CHAR_MAX) {
 printf("International Fractional Digits = %d\n",
 lconv_ptr->int_frac_digits);
}
else {
 printf("International Fractional Digits =");
 printf("[Not available in this locale]\n");
}
}
```

Running the example program produces the following result:

```
International Currency Symbol = GBP
International Fractional Digits = 2
```

---

## localtime, localtime\_r

Convert a time value to broken-down local time.

### Format

```
#include <time.h>

struct tm *localtime (const time_t *timer);

struct tm *localtime_r (const time_t *timer, struct tm *result); (ISO POSIX-1)
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `localtime_r` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Arguments

#### timer

A pointer to a time in seconds since the Epoch. You can generate this time by using the `time` function or you can supply a time.

#### result

A pointer to a `tm` structure where the result is stored. The `tm` structure is defined in the `<time.h>` header file, and is also shown in Table REF-4.

### Description

The `localtime` and `localtime_r` functions convert the time (in seconds since the Epoch) pointed to by *timer* into a broken-down time, expressed as a local time, and store it in a `tm` structure.

The difference between the `localtime_r` and `localtime` functions is that the former stores the result into a user-specified `tm` structure. The latter stores the result into thread-specific static memory allocated by the *Compaq C* RTL, and which is overwritten by subsequent calls to `localtime`; you must make a copy if you want to save it.

On success, `localtime` returns a pointer to the `tm` structure; `localtime_r` returns its second argument. On failure, these functions return the NULL pointer.

The `tm` structure is defined in the `<time.h>` header file and described in Table REF-4.

**Table REF-4 tm Structure**

---

|                           |                                 |
|---------------------------|---------------------------------|
| <code>int tm_sec;</code>  | Seconds after the minute (0-60) |
| <code>int tm_min;</code>  | Minutes after the hour (0-59)   |
| <code>int tm_hour;</code> | Hours since midnight (0-23)     |
| <code>int tm_mday;</code> | Day of the month (1-31)         |
| <code>int tm_mon;</code>  | Months since January (1-11)     |

(continued on next page)

Table REF-4 (Cont.) tm Structure

|                              |                                                                                                                       |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| int tm_year;                 | Years since 1900                                                                                                      |
| int tm_wday;                 | Days since Sunday (0-6)                                                                                               |
| int tm_yday;                 | Days since January 1 (0-365)                                                                                          |
| int tm_isdst;                | Daylight Savings Time flag                                                                                            |
|                              | <ul style="list-style-type: none"><li>tm_isdst = 0 for Standard Time</li><li>tm_isdst = 1 for Daylight Time</li></ul> |
| long tm_gmtoff; <sup>1</sup> | Seconds east of Greenwich (negative values indicate seconds west of Greenwich)                                        |
| char *tm_zone; <sup>1</sup>  | Time zone string, for example "GMT"                                                                                   |

<sup>1</sup>This field is an extension to the ANSI C structure. It is present unless you compile your program with /STANDARD=ANSI89 or with \_DECC\_V4\_SOURCE defined.

The type time\_t is defined in the <time.h> header file as follows:

```
typedef long int time_t
```

**Note**

Generally speaking, UTC-based time functions can affect in-memory time-zone information, which is process-wide data. However, if the system time zone remains the same during the execution of the application (which is the common case) and the cache of timezone files is enabled (which is the default), then the \_r variant of the time functions asctime\_r, ctime\_r, gmtime\_r and localtime\_r, is both thread-safe and AST-reentrant. If, however, the system time zone can change during the execution of the application or the cache of timezone files is not enabled, then both variants of the UTC-based time functions belong to the third class of functions, which are neither thread-safe nor AST-reentrant.

Return Values

|      |                            |
|------|----------------------------|
| x    | Pointer to a tm structure. |
| NULL | Indicates failure.         |



---

## log, log2, log10

Return the logarithm of their arguments.

### Format

```
#include <math.h>

double log (double x);
float logf (float x); (Alpha only)
long double logl (long double x); (Alpha only)
double log2 (double x); (Alpha only)
float log2f (float x); (Alpha only)
long double log2l (long double x); (Alpha only)
double log10 (double x);
float log10f (float x); (Alpha only)
long double log10l (long double x); (Alpha only)
```

### Argument

**x**  
A real number.

### Description

The log functions compute the natural (base e) logarithm of  $x$ .

The log2 functions compute the base 2 logarithm of  $x$ .

The log10 functions compute the common (base 10) logarithm of  $x$ .

### Return Values

|           |                                                                               |
|-----------|-------------------------------------------------------------------------------|
| $x$       | The logarithm of the argument (in the appropriate base).                      |
| −HUGE_VAL | $x$ is 0 (errno is set to ERANGE), or $x$ is negative (errno is set to EDOM). |
| NaN       | $x$ is NaN; errno is set to EDOM.                                             |

## **log1p** *(Alpha only)*

---

## **log1p** *(Alpha only)*

Computes  $\ln(1+y)$  accurately.

### **Format**

```
#include <math.h>

double log1p (double y);
float log1pf (float y);
long double log1pl (long double y);
```

### **Argument**

**y**  
A real number greater than  $-1$ .

### **Description**

The `log1p` functions compute  $\ln(1+y)$  accurately, even for tiny  $y$ .

### **Return Values**

|                  |                                                                                                                 |
|------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>x</b>         | The natural logarithm of $(1+y)$ .                                                                              |
| <b>-HUGE_VAL</b> | $y$ is less than $-1$ (errno is set to <code>EDOM</code> ), or $y = -1$ (errno is set to <code>ERANGE</code> ). |
| <b>NaN</b>       | $y$ is NaN; errno is set to <code>EDOM</code> .                                                                 |

---

**logb** *(Alpha only)*

Returns the radix-independent exponent of the argument.

**Format**

```
#include <math.h>

double logb (double x);
float logbf (float x);
long double logbl (long double x);
```

**Argument**

**x**  
A nonzero, real number.

**Description**

The `logb` functions return the exponent of  $x$ , which is the integral part of  $\log_2 |x|$ , as a signed floating-point value, for nonzero  $x$ .

**Return Values**

|                     |                                                                        |
|---------------------|------------------------------------------------------------------------|
| $x$                 | The exponent of $x$ .                                                  |
| $-\text{HUGE\_VAL}$ | $x = 0.0$ ; <code>errno</code> is set to <code>EDOM</code> .           |
| $+\text{Infinity}$  | $x$ is $+\text{Infinity}$ or $-\text{Infinity}$ .                      |
| $\text{NaN}$        | $y$ is $\text{NaN}$ ; <code>errno</code> is set to <code>EDOM</code> . |

## longjmp

---

## longjmp

Provides a way to transfer control from a nested series of function invocations back to a predefined point without returning normally; that is, by not using a series of return statements. The `longjmp` function restores the context of the environment buffer.

### Format

```
#include <setjmp.h>

void longjmp (jmp_buf env, int value);
```

### Arguments

#### **env**

The environment buffer, which must be an array of integers long enough to hold the register context of the calling function. The type `jmp_buf` is defined in the `<setjmp.h>` header file. The contents of the general-purpose registers, including the program counter (PC), are stored in the buffer.

#### **value**

Passed from `longjmp` to `setjmp`, and then becomes the subsequent return value of the `setjmp` call. If *value* is passed as 0, it is converted to 1.

### Description

When `setjmp` is first called, it returns the value 0. If `longjmp` is then called, naming the same environment as the call to `setjmp`, control is returned to the `setjmp` call as if it had returned normally a second time. The return value of `setjmp` in this second return is the *value* you supply in the `longjmp` call. To preserve the true value of `setjmp`, the function calling `setjmp` must not be called again until the associated `longjmp` is called.

The `setjmp` function preserves the hardware general purpose registers, and the `longjmp` function restores them. After a `longjmp`, all variables have their values as of the time of the `longjmp` except for local automatic variables not marked volatile. These variables have indeterminate values.

The `setjmp` and `longjmp` functions rely on the OpenVMS condition-handling facility to effect a nonlocal goto with a signal handler. The `longjmp` function is implemented by generating a *Compaq C* RTL specified signal and allowing the OpenVMS condition-handling facility to unwind back to the desired destination. The *Compaq C* RTL must be in control of signal handling for any *Compaq C* image.

For *Compaq C* to be in control of signal handling, you must establish all exception handlers through a call to the `VAXC$ESTABLISH` function (rather than `LIB$ESTABLISH`). See Section 4.2.5 and the `VAXC$ESTABLISH` function in this section for more information.

---

#### Note

There are Alpha specific, nonstandard `decc$setjmp` and `decc$fast_longjmp` functions. To use these nonstandard functions instead of the standard ones, a program must be compiled with `__FAST_SETJMP` or `__UNIX_SETJMP` macros defined.

Unlike the standard `longjmp` function, the `decc$fast_longjmp` function does not convert its second argument from 0 to 1. After a call to `decc$fast_longjmp`, a corresponding `setjmp` function returns with the exact value of the second argument specified in the `decc$fast_longjmp` call.

---

## Restrictions

You cannot invoke the `longjmp` function from an OpenVMS condition handler. However, you may invoke `longjmp` from a signal handler that has been established for any signal supported by the *Compaq C* RTL, subject to the following nesting restrictions:

- The `longjmp` function will not work if invoked from nested signal handlers. The result of the `longjmp` function, when invoked from a signal handler that has been entered as a result of an exception generated in another signal handler, is undefined.
- Do not invoke the `setjmp` function from a signal handler unless the associated `longjmp` is to be issued before the handling of that signal is completed.
- Do not invoke the `longjmp` function from within an exit handler (established with `atexit` or `SYS$DCLEXH`). Exit handlers are invoked after image tear-down, so the destination address of the `longjmp` no longer exists.
- Invoking `longjmp` from within a signal handler to return to the main thread of execution might leave your program in an inconsistent state. Possible side effects include the inability to perform I/O or to receive any more UNIX signals.

## longname

---

## longname

Returns the full name of the terminal.

### Format

```
#include <curses.h>
```

```
void longname (char *termbuf, char *name);
```

### Function Variants

The `longname` function has variants named `_longname32` and `_longname64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**termbuf**

A string containing the name of the terminal.

**name**

A character-string buffer with a minimum length of 64 characters.

### Description

The terminal name is in a readable format so that you can double-check to be sure that Curses has correctly identified your terminal. The dummy argument *termbuf* is required for UNIX software compatibility and serves no function in the OpenVMS environment. If portability is a concern, you must write a set of dummy routines to perform the functionality provided by the database `termcap` in the UNIX system environment.

## lrand48

Generates uniformly distributed pseudorandom-number sequences. Returns 48-bit signed long integers.

### Format

```
#include <stdlib.h>

long int lrand48 (void);
```

### Description

The `lrand48` function generates pseudorandom numbers using the linear congruential algorithm and 48-bit integer arithmetic.

It returns nonnegative, long integers uniformly distributed over the range of  $y$  values such that  $0 \leq y < 2^{31}$ .

Before you call the `lrand48` function use either `srand48`, `seed48`, or `lcong48` to initialize the random-number generator. You must initialize prior to invoking the `lrand48` function, because it stores the last 48-bit  $X_i$  generated into an internal buffer. (Although it is not recommended, constant default initializer values are supplied automatically if the `drand48`, `lrand48`, or `mrnd48` functions are called without first calling an initialization function.)

The function works by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

The argument  $m$  equals  $2^{48}$ , so 48-bit integer arithmetic is performed. Unless you invoke the `lcong48` function, the multiplier value  $a$  and the addend value  $c$  are:

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

The value returned by the `lrand48` function is computed by first generating the next 48-bit  $X_i$  in the sequence. Then the appropriate bits, according to the type of data item to be returned, are copied from the high-order (most significant) bits of  $X_i$  and transformed into the returned value.

See also `drand48`, `lcong48`, `mrnd48`, `seed48`, and `srand48` in this section.

### Return Value

|   |                                                                                             |
|---|---------------------------------------------------------------------------------------------|
| n | Signed nonnegative long integers uniformly distributed over the range $0 \leq y < 2^{31}$ . |
|---|---------------------------------------------------------------------------------------------|

---

## lseek

Positions a file to an arbitrary byte position and returns the new position.

### Format

```
#include <unistd.h>

off_t lseek (int file_desc, off_t offset, int direction);
```

### Arguments

#### **file\_desc**

An integer returned by `open`, `creat`, `dup`, or `dup2`.

#### **offset**

The offset, specified in bytes. The `off_t` data type is either a 64-bit integer or a 32-bit integer. The 64-bit interface allows for file sizes greater than 2 gigabytes, and can be selected at compile time by defining the `_LARGEFILE` feature-test macro as follows:

```
CC/DEFINE=_LARGEFILE
```

#### **direction**

An integer indicating whether the offset is to be measured forward from the beginning of the file (`direction=SEEK_SET`), forward from the current position (`direction=SEEK_CUR`), or backward from the end of the file (`direction=SEEK_END`).

### Description

The `lseek` function can position fixed-length record-access file with no carriage control or a stream-access file on any byte offset, but can position all other files only on record boundaries.

The available Standard I/O functions position a record file at its first byte, at the end-of-file, or on a record boundary. Therefore, the arguments given to `lseek` must specify either the beginning or end of the file, a 0 offset from the current position (an arbitrary record boundary), or the position returned by a previous, valid `lseek` call.

This function returns the new file position as an integer of type `off_t` which, like the *offset* parameter, is either a 64-bit integer if `_LARGEFILE` is defined, or a 32-bit integer if not.

For a portable way to position an arbitrary byte location with any type of file, see the `fgetpos` and `fsetpos` functions in this section.

---

#### Caution

---

If, while accessing a stream file, you seek beyond the end-of-file and then write to the file, the `lseek` function creates a hole by filling the skipped bytes with zeros.

In general, for record files, `lseek` should only be directed to an absolute position that was returned by a previous valid call to `lseek` or to the



beginning or end of a file. If a call to `lseek` does not satisfy these conditions, the results are unpredictable.

---

See also `open`, `creat`, `dup`, `dup2`, and `fseek` in this section.

## **Return Values**

|                 |                                                                                                            |
|-----------------|------------------------------------------------------------------------------------------------------------|
| <code>x</code>  | The new file position.                                                                                     |
| <code>-1</code> | Indicates that the file descriptor is undefined, or a seek was attempted before the beginning of the file. |

## lwait

---

## lwait

Waits for I/O on a specific file to complete.

### Format

```
#include <stdio.h>

int lwait (int fd);
```

### Argument

**fd**  
A file descriptor corresponding to an open file.

### Description

The `lwait` function is used primarily to wait for completion of pending asynchronous I/O.

### Return Values

|    |                                  |
|----|----------------------------------|
| 0  | Indicates successful completion. |
| −1 | Indicates an error.              |

---

## malloc

Allocates an area of memory. These functions are AST-reentrant.

### Format

```
#include <stdlib.h>

void *malloc (size_t size);
```

### Function Variants

The malloc function has variants named `_malloc32` and `_malloc64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Argument

#### size

The total number of bytes to be allocated.

### Description

The malloc function allocates a contiguous area of memory whose size, in bytes, is supplied as an argument. The space is not initialized.

---

#### Note

The malloc routines call the system routine LIB\$VM\_MALLOC. Because LIB\$VM\_MALLOC is designed as a general-purpose routine to allocate memory, it is called upon in a wide array of scenarios to allocate and reallocate blocks efficiently. The most common usage is the management of smaller blocks of memory, and the most important aspect of memory allocation under these circumstances is efficiency.

LIB\$VM\_MALLOC makes use of its own free space to satisfy requests, once the heap storage is consumed by splitting large blocks and merging adjacent blocks. Memory can still become fragmented, leaving unused blocks. Once heap storage is consumed, LIB\$VM\_MALLOC manages its own free space and merged blocks to satisfy requests, but varying sizes of memory allocations can cause blocks to be left unused.

Because LIB\$VM\_MALLOC cannot be made to satisfy all situations in the best possible manner, you should perform your own memory management if you have special memory usage needs. This assures the best use of memory for your particular application.

The *OpenVMS Programming Concepts Manual* explains the several memory allocation routines that are available. They are grouped into three levels of hierarchy:

1. At the highest level are the RTL Heap Management Routines LIB\$GET\_VM and LIB\$FREE\_VM, which provide a mechanism for allocating and freeing blocks of memory of arbitrary size. Also at this level are the routines based on the concept of zones, such as LIB\$CREATE\_VM\_ZONE, and so on.

## malloc

2. At the next level are the RTL Page Management routines `LIB$GET_VM_PAGE` and `LIB$FREE_VM_PAGE`, which allocate a specified number of contiguous pages.
  3. At the lowest level are the Memory Management System Services such as `$CRETVA` and `$EXPREG` that provide extensive control over address space allocation. Note that, at this level, you must manage the allocation precisely.
- 

### Return Values

|      |                                                                                                                     |
|------|---------------------------------------------------------------------------------------------------------------------|
| x    | The address of the first byte, which is aligned on a quadword boundary.                                             |
| NULL | Indicates that the function is unable to allocate enough memory. <code>errno</code> is set to <code>ENOMEM</code> . |

---

## mblen

Determines the number of bytes comprising a multibyte character.

### Format

```
#include <stdlib.h>

int mblen (const char *s, size_t n);
```

### Arguments

**s**  
A pointer to the multibyte character.

**n**  
The maximum number of bytes that comprise the multibyte character.

### Description

If the character is  $n$  bytes or less, the `mblen` function returns the number of bytes comprising the multibyte character pointed to by  $s$ . If the character is greater than  $n$  bytes, the function returns  $-1$  to indicate an error.

This function is affected by the `LC_CTYPE` category of the program's current locale.

### Return Values

|      |                                                                                                                        |
|------|------------------------------------------------------------------------------------------------------------------------|
| $x$  | The number of bytes that comprise the multibyte character, if the next $n$ or fewer bytes form a valid character.      |
| $0$  | If $s$ is <code>NULL</code> or a pointer to the <code>NULL</code> character.                                           |
| $-1$ | Indicates an error occurred. The function sets <code>errno</code> to <code>EILSEQ</code> – Invalid character detected. |

---

## mbrlen

Determines the number of bytes comprising a multibyte character.

### Format

```
#include <wchar.h>

size_t mbrlen (const char *s, size_t n, mbstate_t *ps);
```

### Arguments

**s**

A pointer to a multibyte character.

**n**

The maximum number of bytes that comprise the multibyte character.

**ps**

A pointer to the `mbstate_t` object. If a `NULL` pointer is specified, the function uses its internal `mbstate_t` object. `mbstate_t` is an opaque datatype intended to keep the conversion state for the state-dependent codesets.

### Description

The `mbrlen` function is equivalent to the call:

```
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal)
```

Where *internal* is the `mbstate_t` object for the `mbrlen` function.

If the multibyte character pointed to by *s* is of *n* bytes or less, the function returns the number of bytes comprising the character (including any shift sequences).

If either an encoding error occurs or the next *n* bytes contribute to an incomplete but potentially valid multibyte character, the function returns `-1` or `-2`, respectively.

See also `mbrtowc` in this section.

### Return Values

|    |                                                                                                                                                                                                                                  |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | The number of bytes comprising the multibyte character.                                                                                                                                                                          |
| 0  | Indicates that <i>s</i> is a <code>NULL</code> pointer or a pointer to a null byte.                                                                                                                                              |
| -1 | Indicates an encoding error, in which case the next <i>n</i> or fewer bytes do not contribute to a complete and valid multibyte character. <code>errno</code> is set to <code>EILSEQ</code> ; the conversion state is undefined. |
| -2 | Indicates an incomplete but potentially valid multibyte character (all <i>n</i> bytes have been processed).                                                                                                                      |

---

## mbrtowc

Converts a multibyte character to its wide-character representation.

### Format

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

```
size_t mbrtowc (wchar_t *pwc, const char *s, size_t n, mbstate_t *ps);
```

### Arguments

**pwc**

A pointer to the resulting wide-character code.

**s**

A pointer to a multibyte character.

**n**

The maximum number of bytes that comprise the multibyte character.

**ps**

A pointer to the `mbstate_t` object. If a NULL pointer is specified, the function uses its internal `mbstate_t` object. `mbstate_t` is an opaque datatype intended to keep the conversion state for the state-dependent codesets.

### Description

If *s* is a NULL pointer, `mbrtowc` is equivalent to the call:

```
mbrtowc(NULL, "", 1, ps)
```

In this case, the values of *pwc* and *n* are ignored.

If *s* is not a NULL pointer, `mbrtowc` inspects at most *n* bytes beginning with the byte pointed to by *s* to determine the number of bytes needed to complete the next multibyte character (including any shift sequences).

If the function determines that the next multibyte character is completed, it determines the value of the corresponding wide character and then, if *pwc* is not a NULL pointer, stores that value in the object pointed to by *pwc*. If the corresponding wide character is the null wide character, the resulting state described is the initial conversion state.

If `mbrtowc` is called as a counting function, which means that *pwc* is a NULL pointer and *s* is neither a NULL pointer nor a pointer to a null byte, the value of the internal `mbstate_t` object will remain unchanged.

### Return Values

x

The number of bytes comprising the multibyte character.

## **mbrtowc**

- |    |                                                                                                                                                                                                                                              |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | The next <i>n</i> or fewer bytes complete the multibyte character that corresponds to the null wide character (which is the value stored if <i>pwc</i> is not a NULL pointer). The wide-character code corresponding to a null byte is zero. |
| -1 | Indicates an encoding error. The next <i>n</i> or fewer bytes do not contribute to a complete and valid multibyte character. <code>errno</code> is set to <code>EILSEQ</code> . The conversion state is undefined.                           |
| -2 | Indicates an incomplete but potentially valid multibyte character (all <i>n</i> bytes have been processed).                                                                                                                                  |



---

## mbstowcs

Converts a sequence of multibyte characters into a sequence of corresponding wide-character codes.

### Format

```
#include <stdlib.h>
```

```
size_t mbstowcs (wchar_t *pwcs, const char *s, size_t n);
```

### Arguments

#### **pwcs**

A pointer to the array containing the resulting sequence of wide-character codes.

#### **s**

A pointer to the array of multibyte characters.

#### **n**

The maximum number of wide-character codes that can be stored in the array pointed to by *pwcs*.

### Description

The `mbstowcs` function converts a sequence of multibyte characters from the array pointed to by *s* to a sequence of wide-character codes that are stored into the array pointed to by *pwcs*, up to a maximum of *n* codes.

This function is affected by the `LC_CTYPE` category of the program's current locale. If copying takes place between objects that overlap, the behavior is undefined.

### Return Values

*x*

The number of array elements modified or required, not including any terminating zero code. The array will not be zero-terminated if the value returned is *n*. If *pwcs* is the NULL pointer, `mbstowcs` returns the number of elements required for the wide-character array.

`(size_t) -1`

Indicates that an error occurred. The function sets `errno` to `EILSEQ` - Invalid character detected.

---

**mbtowc**

Converts a multibyte character to its wide-character equivalent.

**Format**

```
#include <stdlib.h>

int mbtowc (wchar_t *pwc, const char *s, size_t n);
```

**Arguments****pwc**

A pointer to the resulting wide-character code.

**s**

A pointer to the multibyte character.

**n**

The maximum number of bytes that comprise the next multibyte character.

**Description**

If the character is *n* or fewer bytes, the `mbtowc` function converts the multibyte character pointed to by *s* to its wide-character equivalent. If the character is invalid or greater than *n* bytes, the function returns `-1` to indicate an error.

If *pwc* is a `NULL` pointer and *s* is not a null pointer, the function determines the number of bytes that constitute the multibyte character pointed to by *s* (regardless of the value of *n*).

This function is affected by the `LC_CTYPE` category of the program's current locale.

**Return Values**

|                 |                                                                                                                        |
|-----------------|------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>  | The number of bytes that comprise the valid character pointed to by <i>s</i> .                                         |
| <code>0</code>  | If <i>s</i> is either a <code>NULL</code> pointer or a pointer to the null byte.                                       |
| <code>-1</code> | Indicates an error occurred. The function sets <code>errno</code> to <code>EILSEQ</code> – Invalid character detected. |

---

## mbsinit

Determines whether an `mbstate_t` object describes an initial conversion state.

### Format

```
#include <wchar.h>

int mbsinit (const mbstate_t *ps);
```

### Argument

**ps**

A pointer to the `mbstate_t` object. `mbstate_t` is an opaque datatype intended to keep the conversion state for the state-dependent codesets.

### Description

If *ps* is not a NULL pointer, the `mbsinit` function determines whether the `mbstate_t` object pointed to by *ps* describes an initial conversion state. A zero `mbstate_t` object always describes an initial conversion state.

### Return Values

|         |                                                                                                                                               |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| nonzero | The <i>ps</i> argument is a NULL pointer, or the <code>mbstate_t</code> object pointed to by <i>ps</i> describes an initial conversion state. |
| 0       | The <code>mbstate_t</code> object pointed to by <i>ps</i> does not describe an initial conversion state.                                      |

---

## mbsrtowcs

Converts a sequence of multibyte characters to a sequence of corresponding wide-character codes.

### Format

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

```
size_t mbsrtowcs (wchar_t *dst, const char **src, size_t len, mbstate_t *ps);
```

### Function Variants

The `mbsrtowcs` function has variants named `_mbsrtowcs32` and `_mbsrtowcs64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **dst**

A pointer to the destination array containing the resulting sequence of wide-character codes.

#### **src**

An address of the pointer to an array containing a sequence of multibyte characters to be converted.

#### **len**

The maximum number of wide character codes that can be stored in the array pointed to by *dst*.

#### **ps**

A pointer to the `mbstate_t` object. If a NULL pointer is specified, the function uses its internal `mbstate_t` object. `mbstate_t` is an opaque datatype intended to keep the conversion state for the state-dependent codesets.

### Description

The `mbsrtowcs` function converts a sequence of multibyte characters, beginning in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src*, into a sequence of corresponding wide characters.

If *dst* is not a NULL pointer, the converted characters are stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier for one of the following reasons:

- A sequence of bytes is encountered that does not form a valid multibyte character.
- If *dst* is not a NULL pointer, when *len* codes have been stored into the array pointed to by *dst*.

If *dst* is not a NULL pointer, the pointer object pointed to by *src* is assigned either a NULL pointer (if the conversion stopped because of reaching a terminating null wide character), or the address just beyond the last multibyte character converted (if any). If conversion stopped because of reaching a terminating null wide character, the resulting state described is the initial conversion state.

## Return Values

|           |                                                                                                                                                                               |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>n</i>  | The number of multibyte characters successfully converted, sequence, not including the terminating null (if any).                                                             |
| <i>-1</i> | Indicates an error. A sequence of bytes that do not form valid multibyte character was encountered. <i>errno</i> is set to <i>EILSEQ</i> ; the conversion state is undefined. |

---

## memccpy

Copies characters sequentially between strings in memory areas.

### Format

```
#include <string.h>

void *memccpy (void *dest, void *source, int c, size_t n);
```

### Function Variants

The `memccpy` function has variants named `_memccpy32` and `_memccpy64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**

A pointer to the location of a destination string.

**source**

A pointer to the location of a source string.

**c**

A character that you want to search for.

**n**

The number of character you want to copy.

### Description

The `memccpy` function operates on strings in memory areas. A memory area is a group of contiguous characters bound by a count and not terminated by a null character. The function does not check for overflow of the receiving memory area. The `memccpy` function is defined in the `<string.h>` header file.

The `memccpy` function sequentially copies characters from the location pointed to by *source* into the location pointed to by *dest* until one of the following occurs:

- The character specified by *c* (converted to an unsigned char) is copied.
- The number of characters specified by *n* is copied.

### Return Values

x

A pointer to the character following the character specified by *c* in the string pointed to by *dest*.

NULL

Indicates an error. The character *c* is not found after scanning *n* characters in the string.

---

## memchr

Locates the first occurrence of the specified byte within the initial *size* bytes of a given object.

### Format

```
#include <string.h>

void *memchr (const void *s1, int c, size_t size);
```

### Function Variants

The `memchr` function has variants named `_memchr32` and `_memchr64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**s1**  
A pointer to the object to be searched.

**c**  
The byte value to be located.

**size**  
The length of the object to be searched.  
If *size* is zero, `memchr` returns `NULL`.

### Description

Unlike `strchr`, the `memchr` function does not stop when it encounters a null character.

### Return Values

|                   |                                                                 |
|-------------------|-----------------------------------------------------------------|
| pointer           | A pointer to the first occurrence of the byte.                  |
| <code>NULL</code> | Indicates that the specified byte does not occur in the object. |

## memcmp

---

## memcmp

Compares two objects, byte by byte. The compare operation starts with the first byte in each object.

### Format

```
#include <string.h>

int memcmp (const void *s1, const void *s2, size_t size);
```

### Arguments

**s1**

A pointer to the first object.

**s2**

A pointer to the second object.

**size**

The length of the objects to be compared.

If *size* is zero, the two objects are considered equal.

### Description

The memcmp function uses native byte comparison. The sign of the value returned is determined by the sign of the difference between the values of the first pair of unlike bytes in the objects being compared. Unlike the strcmp function, the memcmp function does not stop when a null character is encountered.

### Return Value

x

An integer less than, equal to, or greater than 0, depending on whether the lexical value of the first object is less than, equal to, or greater than that of the second object.



---

## memcpy

Copies a specified number of bytes from one object to another.

### Format

```
#include <string.h>

void *memcpy (void *dest, const void *source, size_t size);
```

### Function Variants

The `memcpy` function has variants named `_memcpy32` and `_memcpy64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**  
A pointer to the destination object.

**source**  
A pointer to the source object.

**size**  
The length of the object to be copied.

### Description

The `memcpy` function copies *size* bytes from the object pointed to by *source* to the object pointed to by *dest*; it does not check for the overflow of the receiving memory area (*dest*). Unlike the `strcpy` function, the `memcpy` function does not stop when a null character is encountered.

### Return Value

|   |                            |
|---|----------------------------|
| x | The value of <i>dest</i> . |
|---|----------------------------|

---

## memmove

Copies a specified number of bytes from one object to another.

### Format

```
#include <string.h>

void *memmove (void *dest, const void *source, size_t size);
```

### Function Variants

The `memmove` function has variants named `_memmove32` and `_memmove64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**

A pointer to the destination object.

**source**

A pointer to the source object.

**size**

The length of the object to be copied.

### Description

In *Compaq C for OpenVMS Systems*, `memmove` and `memcpy` perform the same function. Programs that require portability should use `memmove` if the area pointed at by *dest* could overlap the area pointed at by *source*.

### Return Value

|   |                            |
|---|----------------------------|
| x | The value of <i>dest</i> . |
|---|----------------------------|

### Example

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

main()
{
 char pdest[14] = "hello there";
 char *psource = "you are there";

 memmove(pdest, psource, 7);
 printf("%s\n", pdest);
}
```

This example produces the following output:

```
you are there
```

---

## memset

Sets a specified number of bytes in a given object to a given value.

### Format

```
#include <string.h>

void *memset (void *s, int value, size_t size);
```

### Function Variants

The `memset` function has variants named `_memset32` and `_memset64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**s**  
An array pointer.

**value**  
The value to be placed in *s*.

**size**  
The number of bytes to be placed in *s*.

### Description

The `memset` function copies *value* (converted to an unsigned char) into each of the first *size* characters of the object pointed to by *s*.

This function returns *s*. It does not check for the overflow of the receiving memory area pointed to by *s*.

### Return Value

|   |                         |
|---|-------------------------|
| x | The value of <i>s</i> . |
|---|-------------------------|

---

## mkdir

Creates a directory.

### Format

```
#include <stat.h>
```

```
int mkdir (const char *dir_spec, mode_t mode); (ISO POSIX-1)
```

```
int mkdir (const char *dir_spec, mode_t mode, ...); (COMPAQ C Extension)
```

### Arguments

#### **dir\_spec**

A valid OpenVMS or UNIX style directory specification that may contain a device name. For example:

```
DBA0:[BAY.WINDOWS] /* OpenVMS */
/dba0/bay/windows /* UNIX style */
```

This specification cannot contain a node name, file name, file extension, file version, or a wildcard character. The same restriction applies to the UNIX style directory specifications. For more information about the restrictions on UNIX style specifications, see Chapter 1.

#### **mode**

A file protection. See the `chmod` function in this section for information about the specific file protections.

The file protection of the new directory is derived from the *mode* argument, the process's file protection mask (see the `umask` function), and the parent-directory default protections.

In a manner consistent with the OpenVMS behavior for creating directories, `mkdir` never applies delete access to the directory. An application that needs to set delete access should use an explicit call to `chmod` to set write permission.

See the Description section of this function for more information about how the file protection is set for the newly created directory.

...

Represents the following optional arguments. These arguments have fixed position in the argument list, and cannot be arbitrarily placed.

#### unsigned int **uic**

The user identification code (UIC) that identifies the owner of the created directory. If this argument is 0, the *Compaq C* RTL gives the created directory the UIC of the parent directory. If this argument is not specified, the *Compaq C* RTL gives the created directory your UIC. This optional argument is specific to the *Compaq C* RTL and is not portable.

#### unsigned short **max\_versions**

The maximum number of file versions to be retained in the created directory. The system automatically purges the directory keeping, at most, *max\_versions* number of every file.

If this argument is 0, the *Compaq C* RTL does not place a limit on the maximum number of file versions.

If this argument is not specified, the *Compaq C* RTL gives the created directory the default version limit of the parent directory.

This optional argument is specific to the *Compaq C* RTL and is not portable.

unsigned short **r\_v\_number**

The volume (device) on which to place the created directory if the device is part of a volume set. If this argument is not specified, the *Compaq C* RTL arbitrarily places the created directory within the volume set. This optional argument is specific to the *Compaq C* RTL and is not portable.

## Description

If *dir\_spec* specifies a path that includes directories, which do not exist, intermediate directories are also created. This differs from the behavior of the UNIX system where these intermediate directories must exist and will not be created.

If you do not specify any optional arguments, the *Compaq C* RTL gives the directory your UIC and the default version limit of the parent directory, and arbitrarily places the directory within the volume set. You cannot get the default behavior for the *uic* or *max\_versions* arguments if you specify any arguments after them.

---

### Note

---

The way to create files with OpenVMS RMS default protections using the UNIX system-call functions `umask`, `mkdir`, `creat`, and `open` is to call `mkdir`, `creat`, and `open` with a file-protection mode argument of 0777 in a program that never specifically calls `umask`. These default protections include correctly establishing protections based on ACLs, previous versions of files, and so on.

In programs that do `vfork/exec` calls, the new process image inherits whether `umask` has ever been called or not from the calling process image. The `umask` setting and whether the `umask` function has ever been called are both inherited attributes.

---

The file protection supplied by the *mode* argument is modified by the process's file protection mask in such a way that the file protection for the new directory is set to the bitwise AND of the *mode* argument and the complement of the file protection mask.

Default file protections are supplied to the new directory from the parent-directory such that if a protection value bit in the new directory is zero, then the value of this bit is inherited from the parent-directory. However, bits in the parent directory's file protection that indicate delete access do not cause corresponding bits to be set in the new directory's file protection.

## Return Values

|    |                    |
|----|--------------------|
| 0  | Indicates success. |
| -1 | Indicates failure. |

## mkdir

### Examples

1. 

```
umask (0002); /* turn world write access off */
mkdir ("sys$disk:[.parentdir.childdir]", 0222); /* turn write
 access on */
```

Parent directory file protection: System:RWD, Owner:RWD, Group:R,  
World:R

The file protection derived from the combination of the mode argument and the file protection mask set by umask is (0222) & ~(0002), which is 0220. When the parent directory defaults are applied to this protection, the protection for the new directory becomes:

File protection:     System:RWD, Owner:RWD, Group:RWD, World:R

2. 

```
umask (0000);
mkdir ("sys$disk:[.parentdir.childdir]", 0444); /* turn read
 access on */
```

Parent directory file protection: System:RWD, Owner:RWD,  
Group:RWD, World:RWD

The file protection derived from the combination of the mode argument and the file protection mask set by umask is (0444) & ~(0000) which is 0444. When the parent directory defaults are applied to this protection, the protection for the new directory is:

File protection:     System:RW, Owner:RW, Group:RW, World:RW

Note that delete access is not inherited.

---

## mkstemp

Constructs a unique file name.

### Format

```
#include <stdlib.h>

int mkstemp (char *template);
```

### Argument

#### **template**

A pointer to a string that is replaced with a unique file name. The string in the *template* argument must be a file name with six trailing Xs.

### Description

The `mkstemp` function replaces the six trailing Xs of the string pointed to by *template* with a unique set of characters, and returns a file descriptor for the file open for reading and writing.

The string pointed to by *template* should look like a file name with six trailing X's. The `mkstemp` function replaces each X with a character from the portable file-name character set, making sure not to duplicate an existing file name.

If the string pointed to by *template* does not contain six trailing Xs, `-1` is returned.

### Return Values

|    |                                                                                                  |
|----|--------------------------------------------------------------------------------------------------|
| x  | An open file descriptor.                                                                         |
| -1 | Indicates an error. (The string pointed to by <i>template</i> does not contain six trailing Xs.) |

## mktemp

---

## mktemp

Creates a unique file name from a template.

### Format

```
#include <stdlib.h>

char *mktemp (char *template);
```

### Function Variants

The `mktemp` function has variants named `_mktemp32` and `_mktemp64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Argument

#### **template**

A pointer to a buffer containing a user-defined template. You supply the template in the form, `namXXXXXX`. The six trailing Xs are replaced by a unique series of characters. You may supply the first three characters. Because the template argument is overwritten, do not specify a string literal (const object).

### Description

The use of `mktemp` is not recommended for new applications. See the `tmpnam` and `mkstemp` functions for the preferable alternatives.

### Return Value

|   |                                                                                                                                                                                              |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x | A pointer to the template, with the template modified to contain the created file name. If this value is a pointer to a null string, it indicates that a unique file name cannot be created. |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



---

## mktime

Converts a local-time structure into time since the Epoch.

### Format

```
#include <time.h>

time_t mktime (struct tm *timeptr);
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `mktime` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Argument

#### **timeptr**

A pointer to the local-time structure.

### Description

The `mktime` function converts the local-time structure, pointed to by *timeptr*, to a time in seconds since the Epoch in the same manner as the values returned by the `time` function. If the local time cannot be encoded, then `mktime` returns the value `(time_t)(-1)`.

The `time_t` type is defined in the `<time.h>` header file as follows:

```
typedef unsigned long int time_t;
```

Local time-zone information is set as if `mktime` called `tzset`.

If the `tm_isdst` field in the local-time structure pointed to by *timeptr* is positive, `mktime` initially presumes that Daylight Savings Time (DST) is in effect for the specified time.

If `tm_isdst` is 0, `mktime` initially presumes that DST is not in effect.

If `tm_isdst` is negative, `mktime` attempts to determine whether or not DST is in effect for the specified time.

### Return Values

|                           |                                                                                                                                                                          |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>            | The specified calendar time encoded as a value of type <code>time_t</code> .                                                                                             |
| <code>(time_t)(-1)</code> | <p>If the local time cannot be encoded.</p> <p>Be aware that a return value of <code>(time_t)(-1)</code> can also represent the valid date: Sun Feb 7 06:28:15 2106.</p> |

## mmap

---

## mmap

Maps file system object into virtual memory.

### Format

```
#include <types.h>
#include <mman.h>

void mmap (void *addr, size_t len, int prot, int flags, int fildes, off_t off); (X/Open, POSIX-1)
void mmap (void *addr, size_t len, int prot, int flags, int fildes, off_t off ...); (COMPAQ C Extension)
```

### Function Variants

The `mmap` function has variants named `_mmap32` and `_mmap64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **addr**

The starting address of the new region (truncated to a page boundary).

#### **len**

The length in bytes of the new region (rounded up to a page boundary).

#### **prot**

Access permission, as defined in the `<mman.h>` header file. Specify either `PROT_NONE`, `PROT_READ`, or `PROT_WRITE`.

#### **flags**

Attributes of the mapped region as the results of a bitwise-inclusive OR operation on any combination of the following:

- `MAP_FILE` or `MAP_ANONYMOUS`
- `MAP_VARIABLE` or `MAP_FIXED`
- `MAP_SHARED` or `MAP_PRIVATE`

#### **fildes**

The file that you want to map to the new mapped file region returned by the `open` function.

#### **off**

The offset, specified in bytes. The `off_t` data type is either a 64-bit integer or a 32-bit integer. The 64-bit interface allows for file sizes greater than 2 gigabytes, and can be selected at compile time by defining the `_LARGEFILE` feature-test macro as follows:

```
CC/DEFINE=_LARGEFILE
```

...

An optional integer specifying additional flags for the `SYS$CRMPSC` system service for `MAP_SHARED`. This optional argument (*Compaq C Extension*) of the `mmap` function was introduced in OpenVMS Version 7.2.

## Description

The `mmap` function creates a new mapped file region, a new private region, or a new shared memory region.

Your application must ensure correct synchronization when using `mmap` in conjunction with any other file access method, such as `read` and `write`, and standard input/output.

Before calling `mmap`, the calling application must also ensure that all bytes in the range `[off, off+len]` are written to the file (using the `fsync` function, for example). If this requirement is not met, `mmap` fails with `errno` set to `ENXIO` (No such device or address).

The `addr` and `len` arguments specify the requested starting address and length in bytes for the new region. The address is a multiple of the page size returned by `sysconf(_SC_PAGE_SIZE)`.

If the `len` argument is not a multiple of the page size returned by `sysconf(_SC_PAGE_SIZE)`, then the result of any reference to an address between the end of the region and the end of the page containing the end of the region is undefined.

The `flags` argument specifies attributes of the mapped region. Values for `flags` are constructed by a bitwise-inclusive OR operation on the flags from the following list of symbolic names defined in the `<mman.h>` header file:

|                            |                                       |
|----------------------------|---------------------------------------|
| <code>MAP_FILE</code>      | Create a mapped file region.          |
| <code>MAP_ANONYMOUS</code> | Create an unnamed memory region.      |
| <code>MAP_VARIABLE</code>  | Place region at the computed address. |
| <code>MAP_FIXED</code>     | Place region at fixed address.        |
| <code>MAP_SHARED</code>    | Share changes.                        |
| <code>MAP_PRIVATE</code>   | Changes are private.                  |

The `MAP_FILE` and `MAP_ANONYMOUS` flags control whether the region you want to map is a mapped file region or an anonymous shared memory region. One of these flags must be selected.

If `MAP_FILE` is set in the `flags` argument:

- A new mapped file region is created, mapping the file associated with the `filedes` argument.
- The `off` argument specifies the file byte offset where the mapping starts. This offset must be a multiple of the page size returned by `sysconf(_SC_PAGE_SIZE)`.
- If the end of the mapped file region is beyond the end of the file, the result of any reference to an address in the mapped file region corresponding to an offset beyond the end of the file is unspecified.

If `MAP_ANONYMOUS` is set in the `flags` argument:

- A new memory region is created and initialized to all zeros.
- If the `filedes` argument is not `-1`, the `mmap` function fails.

## mmap

The new region is placed at the requested address if the requested address is not null and it is possible to place the region at this address. When the requested address is null or the region cannot be placed at the requested address, the `MAP_VARIABLE` and `MAP_FIXED` flags control the placement of the region. One of these flags must be selected.

If `MAP_VARIABLE` is set in the *flags* argument:

- If the requested address is null or if it is not possible for the system to place the region at the requested address, the region is placed at an address selected by the system.

If `MAP_FIXED` is set in the *flags* argument:

- If the requested address is not null, the `mmap` function succeeds even if the requested address is already part of another region. (If the address is within an existing region, the effect on the pages within that region and within the area of the overlap produced by the two regions is the same as if they were unmapped. In other words, whatever is mapped between *addr* and *addr + len* is unmapped.)
- If the requested address is null and `MAP_FIXED` is specified, the results are undefined.

The `MAP_PRIVATE` and `MAP_SHARED` flags control the visibility of modifications to the mapped file or shared memory region. One of these flags must be selected.

If `MAP_SHARED` is set in the *flags* argument:

- If the region is a mapped region, modifications to the region are visible to other processes that mapped the same region using `MAP_SHARED`.
- If the region is a mapped file region, modifications to the region are written to the file. (Note that the modifications are not immediately written to the file because of buffer cache delay; that is, the write to the file does not occur until there is a need to reuse the buffer cache. If the modifications must be written to the file immediately, use the `msync` function to ensure that this is done.)

If `MAP_PRIVATE` is set in the *flags* argument:

- Modifications to the mapped region by the calling process are not visible to other processes that mapped the same region using either `MAP_PRIVATE` or `MAP_SHARED`.
- Modifications to the mapped region by the calling process are not written to the file.

It is unspecified whether modifications by processes that mapped the region using `MAP_SHARED` are visible to other processes that mapped the same region using `MAP_PRIVATE`.

The *prot* argument specifies access permissions for the mapped region. Specify one of the following:

|                         |                   |
|-------------------------|-------------------|
| <code>PROT_NONE</code>  | No access         |
| <code>PROT_READ</code>  | Read-only         |
| <code>PROT_WRITE</code> | Read/Write access |

After the successful completion of the `mmap` function, you can close the *filedes* argument without effect on the mapped region or on the contents of the mapped file. Each mapped region creates a file reference, similar to an open file descriptor, that prevents the file data from being deallocated.

---

**Note**

---

The following rules apply to OpenVMS specific file references:

- Because of the additional file reference, if *filedes* is not opened for file sharing, `mmap` reopens it with file sharing enabled.
  - The additional file reference that remains for mapped regions implies that a later `open`, `fopen`, or `create` call to the file that is mapped must specify file sharing.
- 

Modifications made to the file using the `write` function are visible to mapped regions, and modifications to a mapped region are visible with the `read` function.

---

**Note**

---

Beginning with OpenVMS Version 7.2, while processing a `MAP_SHARED` request, the `mmap` function constructs the *flags* argument of the `SYS$CRMPSC` service as a bitwise inclusive OR of those bits it sets by itself to fulfill the `MAP_SHARED` request and those bits specified by the caller in the optional argument.

By default, for `MAP_SHARED` the `mmap` function creates a temporary group global section. The optional `mmap` argument provides the caller with direct access to the features of the `SYS$CRMPSC` system service.

Using the optional argument, the caller can create, for example, a system global section (`SEC$M_SYSGBL` bit) or permanent global section (`SEC$M_PERM` bit). For example, to create a system permanent global section, the caller can specify `(SEC$M_SYSGBL | SEC$M_PERM)` in the optional argument.

The `mmap` function does not check or set any privileges. It is the responsibility of the caller to set appropriate privileges, such as `SYSGBL` privilege for `SEC$M_SYSGBL`, and `PRMGBL` for `SEC$M_PERM`, before calling `mmap` with the optional argument.

---

See also `read`, `write`, `open`, `fopen`, `creat`, and `sysconf` in this section.

## Return Values

|   |                                          |
|---|------------------------------------------|
| x | The address where the mapping is placed. |
|---|------------------------------------------|

## mmap

### MAP\_FAILED

Indicates an error; `errno` is set to one of the following values:

- `EACCES` – The file referred to by *filedes* is not open for read access, or the file is not open for write access and `PROT_WRITE` was set for a `MAP_SHARED` mapping operation.
- `EBADF` – The *filedes* argument is not a valid file descriptor.
- `EINVAL` – The *flags* or *prot* argument is invalid, or the *addr* argument or *off* argument is not a multiple of the page size returned by `sysconf(_SC_PAGE_SIZE)`. Or `MAP_ANONYMOUS` was specified in *flags* and *filedes* is not `-1`.
- `ENODEV` – The file descriptor *filedes* refers to an object that cannot be mapped, such as a terminal.
- `ENOMEM` – There is not enough address space to map *len* bytes.
- `ENXIO` – The addresses specified by the range `[off, off + len]` are invalid for *filedes*.
- `EFAULT` – The *addr* argument is an invalid address.

---

## modf

Decomposes a floating-point number.

### Format

```
#include <math.h>

double modf (double x, double *iptr);
float modff (float x, float *iptr); (Alpha only)
long double modfl (long double x, long double *iptr); (Alpha only)
```

### Arguments

**x**

An object of type double, float, or long double.

**iptr**

A pointer to an object of type double, float, or long double to match the type of *x*.

### Description

The modf functions decompose their first argument *x* into a positive fractional part *f* and an integer part *i*, each of which has the same sign as *x*.

The functions return *f* and assign *i* to the object pointed to by the second argument (*iptr*).

### Return Values

|          |                                                                        |
|----------|------------------------------------------------------------------------|
| <b>x</b> | The fractional part of the argument <i>x</i> .                         |
| NaN      | <i>x</i> is NaN; errno is set to EDOM and * <i>iptr</i> is set to NaN. |
| 0        | Underflow occurred; errno is set to ERANGE.                            |

## [w]move

---

## [w]move

Change the current cursor position on the specified window to the coordinates (y,x). The move function acts on the stdscr window.

### Format

```
#include <curses.h>

int move (int y, int x);
int wmove (WINDOW *win, int y, int x);
```

### Arguments

**win**  
A pointer to the window.

**y**  
A window coordinate.

**x**  
A window coordinate.

### Description

For more information, see the `scrollok` function in this section.

### Return Values

|     |                                                                |
|-----|----------------------------------------------------------------|
| OK  | Indicates success.                                             |
| ERR | Indicates that the function makes the screen scroll illegally. |



---

## mprotect

Modifies access protections of memory mapping.

### Format

```
#include <mman.h>

int mprotect (void *addr, size_t len, int prot);
```

### Arguments

**addr**

The address of the region that you want to modify.

**len**

The length in bytes of the region that you want to modify.

**prot**

Access permission, as defined in the `<mman.h>` header file. Specify either `PROT_NONE`, `PROT_READ`, or `PROT_WRITE`.

### Description

The `mprotect` function modifies the access protection of a mapped file or shared memory region.

The *addr* and *len* arguments specify the address and length in bytes of the region that you want to modify. The *len* argument must be a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`. If *len* is not a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`, the length of the region is rounded up to the next multiple of the page size.

The *prot* argument specifies access permissions for the mapped region. Specify one of the following:

|                         |                   |
|-------------------------|-------------------|
| <code>PROT_NONE</code>  | No access         |
| <code>PROT_READ</code>  | Read-only         |
| <code>PROT_WRITE</code> | Read/Write access |

The `mprotect` function does not modify the access permission of any region that lies outside of the specified region, except that the effect on addresses between the end of the region, and the end of the page containing the end of the region, is unspecified.

If the `mprotect` function fails under a condition other than that specified by `EINVAL`, the access protection of some of the pages in the range [*addr*, *addr* + *len*] can change. Suppose the error occurs on some page at an *addr2*; `mprotect` can modify protections of all whole pages in the range [*addr*, *addr2*].

See also `sysconf` in this section.

## mprotect

### Return Values

0

Indicates success.

−1

Indicates an error; `errno` is set to one of the following values:

- `EACCESS` – The *prot* argument specifies a protection that conflicts with the access permission set for the underlying file.
- `EINVAL` – The *prot* argument is invalid, or the *addr* argument is not a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`.
- `EFAULT` – The range [*addr*, *addr* + *len*] includes an invalid address.

## mrnd48

Generates uniformly distributed pseudorandom-number sequences. Returns 48-bit signed long integers.

### Format

```
#include <stdlib.h>

long int mrnd48 (void);
```

### Description

The `mrnd48` function generates pseudorandom numbers using the linear congruential algorithm and 48-bit integer arithmetic.

It returns signed long integers uniformly distributed over the range of  $y$  values such that  $-2^{31} \leq y < 2^{31}$ .

Before you call the `mrnd48` function, use either `srand48`, `seed48`, or `lcong48` to initialize the random-number generator. You must initialize the `mrnd48` function prior to invoking it, because it stores the last 48-bit  $X_i$  generated into an internal buffer. (Although it is not recommended, constant default initializer values are supplied automatically if the `drand48`, `lrand48`, or `mrnd48` functions are called without first calling an initialization function.)

The function works by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

The argument  $m$  equals  $2^{48}$ , so 48-bit integer arithmetic is performed. Unless you invoke the `lcong48` function, the multiplier value  $a$  and the addend value  $c$  are:

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

The values returned by the `mrnd48` function is computed by first generating the next 48-bit  $X_i$  in the sequence. Then the appropriate bits, according to the type of returned data item, are copied from the high-order (most significant) bits of  $X_i$  and transformed into the returned value.

See also `drand48`, `lrand48`, `lcong48`, `seed48`, and `srand48` in this section.

### Return Values

|   |                                                                                               |
|---|-----------------------------------------------------------------------------------------------|
| n | Returns signed long integers uniformly distributed over the range $-2^{31} \leq y < 2^{31}$ . |
|---|-----------------------------------------------------------------------------------------------|

---

## msync

Synchronizes a mapped file.

### Format

```
#include <mman.h>

int msync (void *addr, size_t len, int flags);
```

### Arguments

#### addr

The address of the region that you want to synchronize.

#### len

The length in bytes of the region that you want to synchronize.

#### flags

One of the following symbolic constants defined in the `<mman.h>` header file:

|               |                          |
|---------------|--------------------------|
| MS_SYNC       | Synchronous cache flush  |
| MS_ASYNC      | Asynchronous cache flush |
| MS_INVALIDATE | Invalidate cached pages  |

### Description

The `msync` function controls the caching operations of a mapped file region. Use `msync` to:

- Ensure that modified pages in the region transfer to the underlying storage device of the file.
- Control the visibility of modifications with respect to file system operations.

The *addr* and *len* arguments specify the region to be synchronized.

The *len* argument must be a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`; otherwise, the length of the region is rounded up to the next multiple of the page size.

If the *flags* argument is set to:

| <i>flags</i> Argument | Then the <code>msync</code> Function...                                                                                                                                  |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MS_SYNC               | Does not return until the system completes all I/O operations.                                                                                                           |
| MS_ASYNC              | Returns after the system schedules all I/O operations.                                                                                                                   |
| MS_INVALIDATE         | Invalidates all cached copies of the pages. The operating system must obtain new copies of the pages from the file system the next time the application references them. |

After a successful call to the `msync` function with the *flags* argument set to:

- MS\_SYNC – All previous modifications to the mapped region are visible to processes using the `read` argument. Previous modifications to the file using the `write` function are lost.

- `MS_INVALIDATE` – All previous modifications to the file using the `write` function are visible to the mapped region. Previous direct modifications to the mapped region are lost.

See also `read`, `write`, and `sysconf` in this section.

## Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| −1 | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"><li>• <code>EIO</code> – An I/O error occurred while reading from or writing to the file system.</li><li>• <code>ENOMEM</code> – The range specified by <code>[addr, addr + len]</code> is invalid for a process' address space, or the range specifies one or more unmapped pages.</li><li>• <code>EINVAL</code> – The <code>addr</code> argument is not a multiple of the page size as returned by <code>sysconf(_SC_PAGE_SIZE)</code>.</li><li>• <code>EFAULT</code> – The range <code>[addr, addr + len]</code> includes an invalid address.</li></ul> |

## **munmap**

---

## **munmap**

Unmaps a mapped region.

### **Format**

```
#include <mman.h>

int munmap (void *addr, size_t len);
```

### **Arguments**

#### **addr**

The address of the region that you want to unmap.

#### **len**

The length in bytes of that region the you want to unmap.

### **Description**

The `munmap` function unmaps a mapped file or shared memory region.

The *addr* and *len* arguments specify the address and length in bytes, respectively, of the region to be unmapped.

The *len* argument must be a multiple of the page size as returned by `sysconf(_SC_PAGE_SIZE)`; otherwise, the length of the region is rounded up to the next multiple of the page size.

The result of using an address that lies in an unmapped region and not in any subsequently mapped region is undefined.

See also `sysconf` in this section.

### **Return Values**

|    |                                                                                                                                                                                                                                                                                                                                                                                      |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                                                                                                                                                                                                                                                                   |
| -1 | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"><li>• <code>EINVAL</code> – The <i>addr</i> argument is not a multiple of the page size as returned by <code>sysconf(_SC_PAGE_SIZE)</code>.</li><li>• <code>EFAULT</code> – The range [<i>addr</i>, <i>addr</i> + <i>len</i>] includes an invalid address.</li></ul> |

---

## mv[w]addch

Move the cursor to coordinates (y,x) and add a character to the specified window.

### Format

```
#include <curses.h>

int mvaddch (int y, int x, char ch);
int mvwaddch (WINDOW *win, int y, int x, char ch);
```

### Arguments

**win**

A pointer to the window.

**y**

A window coordinate.

**x**

A window coordinate.

**ch**

If this argument is a new-line character (`\n`), the `mvaddch` and `mvwaddch` functions clear the line to the end, and move the cursor to the next line at the same x coordinate. A carriage return (`\r`) moves the cursor to the beginning of the specified line. A tab (`\t`) moves the cursor to the next tabstop within the window.

### Description

This routine performs the same function as `mvwaddch`, but on the `stdscr` window.

When `mvwaddch` is used on a subwindow, it writes the character onto the underlying window as well.

### Return Values

OK

Indicates success.

ERR

Indicates that writing the character would cause the screen to scroll illegally. For more information, see the `scrollok` function in this section.

## mv[w]addstr

---

## mv[w]addstr

Move the cursor to coordinates (y,x) and add the specified string, to which *str* points, to the specified window.

### Format

```
#include <curses.h>

int mvaddstr (int y, int x, char *str);
int mvwaddstr (WINDOW *win, int y, int x, char *str);
```

### Arguments

**win**

A pointer to the window.

**y**

A window coordinate.

**x**

A window coordinate.

**str**

A pointer to the character string.

### Description

This routine performs the same function as `mvwaddstr`, but on the `stdscr` window.

When `mvwaddstr` is used on a subwindow, the string is written onto the underlying window as well.

### Return Values

OK

Indicates success.

ERR

Indicates that the function causes the screen to scroll illegally, but it places as much of the string onto the window as possible. For more information, see the `scrollok` function in this section.



---

## mvcur

Moves the terminal's cursor from *(lasty,lastx)* to *(newy,newx)*.

### Format

```
#include <curses.h>

int mvcur (int lasty, int lastx, int newy, int newx);
```

### Arguments

**lasty**  
The cursor position.

**lastx**  
The cursor position.

**newy**  
The resulting cursor position.

**newx**  
The resulting cursor position.

### Description

In *Compaq C* for *OpenVMS* Systems, `mvcur` and `move` perform the same function.  
See also `move` in this section.

### Return Values

|     |                                                                                                                                            |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------|
| OK  | Indicates success.                                                                                                                         |
| ERR | Indicates that moving the window put part or all of the window off the edge of the terminal screen. The terminal screen remains unaltered. |

## mv[w]delch

---

## mv[w]delch

Move the cursor to coordinates (y,x) and delete the character on the specified window. The mvdelch function acts on the stdscr window.

### Format

```
#include <curses.h>

int mvdelch (int y, int x);
int mvwdelch (WINDOW *win, int y, int x);
```

### Arguments

**win**

A pointer to the window.

**y**

A window coordinate.

**x**

A window coordinate.

### Description

Each of the following characters on the same line shifts to the left, and the last character becomes blank.

### Return Values

OK

Indicates success.

ERR

Indicates that deleting the character would cause the screen to scroll illegally. For more information, see the `scrollok` function in this section.

---

## mv[w]getch

Move the cursor to coordinates (y,x), get a character from the terminal screen, and echo it on the specified window. The mvgetch function acts on the stdscr window.

### Format

```
#include <curses.h>

int mvgetch (int y, int x);
int mvwgetch (WINDOW *win, int y, int x);
```

### Arguments

**win**  
A pointer to the window.

**y**  
A window coordinate.

**x**  
A window coordinate.

### Description

The mvgetch and mvwgetch functions refresh the specified window before fetching the character.

### Return Values

|     |                                                                                                                                     |
|-----|-------------------------------------------------------------------------------------------------------------------------------------|
| x   | The returned character.                                                                                                             |
| ERR | Indicates that the function causes the screen to scroll illegally. For more information, see the scrollok function in this section. |

## mv[w]getstr

---

## mv[w]getstr

Move the cursor to coordinates (*y,x*), get a string from the terminal screen, store it in the variable *str* (which must be large enough to contain the string), and echo it on the specified window. The *mvgetstr* function acts on the *stdscr* window.

### Format

```
#include <curses.h>

int mvgetstr (int y, int x, char *str);

int mvwgetstr (WINDOW *win, int y, int x, char *str);
```

### Arguments

**win**

A pointer to the window.

**y**

A window coordinate.

**x**

A window coordinate.

**str**

The string that is displayed.

### Description

The *mvgetstr* and *mvwgetstr* functions strip the new-line terminator (`\n`) from the string.

### Return Values

OK

Indicates success.

ERR

Indicates that the function causes the screen to scroll illegally.

---

## mv[w]inch

Move the cursor to coordinates (y,x) and return the character on the specified window without making changes to the window. The mvinch function acts on the stdscr window.

### Format

```
#include <curses.h>

int mvinch (int y, int x);

int mvwinch (WINDOW *win, int y, int x);
```

### Arguments

**win**  
A pointer to the window.

**y**  
A window coordinate.

**x**  
A window coordinate.

### Return Values

|     |                           |
|-----|---------------------------|
| x   | The returned character.   |
| ERR | Indicates an input error. |

## mv[w]insch

---

## mv[w]insch

Move the cursor to coordinates  $(y,x)$  and insert the character *ch* into the specified window. The mvinsch function acts on the stdscr window.

### Format

```
#include <curses.h>

int mvinsch (int y, int x, char ch);

int mvwinsch (WINDOW *win, int y, int x, char ch);
```

### Arguments

**win**

A pointer to the window.

**y**

A window coordinate.

**x**

A window coordinate.

**ch**

The character to be inserted at the window's coordinates.

### Description

After the character is inserted, each character on the line shifts to the right, and the last character on the line is deleted.

### Return Values

OK

Indicates success.

ERR

Indicates that the function makes the screen scroll illegally. For more information, see the scrollok function in this section.

---

## mv[w]insstr

Move the cursor to coordinates (y,x) and insert the specified string into the specified window. The mvinsstr function acts on the stdscr window.

### Format

```
#include <curses.h>

int mvinsstr (int y, int x, char *str);

int mvwinsstr (WINDOW *win, int y, int x, char *str);
```

### Arguments

**win**

A pointer to the window.

**y**

A window coordinate.

**x**

A window coordinate.

**str**

The string that is displayed.

### Description

Each character after the string shifts to the right, and the last character disappears. The mvinsstr and mvwinsstr functions are specific to *Compaq C* for *OpenVMS* Systems and are not portable.

### Return Values

OK

Indicates success.

ERR

Indicates that the function makes the screen scroll illegally. For more information, see the scrollok function in this section.

## mvwin

---

## mvwin

Moves the starting position of the window to the specified (y,x) coordinates.

### Format

```
#include <curses.h>

mvwin (WINDOW *win, int y, int x);
```

### Arguments

**win**  
A pointer to the window.

**y**  
A window coordinate.

**x**  
A window coordinate.

### Description

When moving subwindows, the `mvwin` function does not rewrite the contents of the subwindow on the underlying window at the new position. If you write anything to the subwindow after the move, the function also writes to the underlying window.

### Return Values

|     |                                                                                                                                            |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------|
| OK  | Indicates success.                                                                                                                         |
| ERR | Indicates that moving the window put part or all of the window off the edge of the terminal screen. The terminal screen remains unaltered. |



---

## newwin

Creates a new window with *numlines* lines and *numcols* columns starting at the coordinates (*begin\_y*,*begin\_x*) on the terminal screen.

### Format

```
#include <curses.h>
```

```
WINDOW *newwin (int numlines, int numcols, int begin_y, int begin_x);
```

### Arguments

#### **numlines**

If it is 0, the newwin function sets that dimension to LINES (*begin\_y*). To get a new window of dimensions LINES by COLS, use the following line:

```
newwin (0, 0, 0, 0)
```

#### **numcols**

If it is 0, the newwin function sets that dimension to COLS (*begin\_x*). Thus, to get a new window of dimensions LINES by COLS, use the following line:

```
newwin (0, 0, 0, 0)
```

#### **begin\_y**

A window coordinate.

#### **begin\_x**

A window coordinate.

### Return Values

|     |                                      |
|-----|--------------------------------------|
| x   | The address of the allocated window. |
| ERR | Indicates an error.                  |

## nextafter *(Alpha only)*

---

## nextafter *(Alpha only)*

Returns the next machine-representable number following  $x$  in the direction of  $y$ .

### Format

```
#include <math.h>

double nextafter (double x, double y);
float nextafterf (float x, float y);
long double nextafterl (long double x, long double y);
```

### Arguments

**x**  
A real number.

**y**  
A real number.

### Description

The `nextafter` functions return the next machine-representable floating-point number following  $x$  in the direction of  $y$ . If  $y$  is less than  $x$ , `nextafter` returns the largest representable floating-point number less than  $x$ .

### Return Values

|                       |                                                                                     |
|-----------------------|-------------------------------------------------------------------------------------|
| <code>x</code>        | The next representable floating-point value following $x$ in the direction of $y$ . |
| <code>HUGE_VAL</code> | Overflow; <code>errno</code> is set to <code>ERANGE</code> .                        |
| <code>NaN</code>      | $x$ or $y$ is <code>NaN</code> ; <code>errno</code> is set to <code>EDOM</code> .   |

---

## nice

Increases or decreases process priority relative to the process current priority by the amount of the argument. This function is nonreentrant.

### Format

```
#include <unistd.h>

int nice (int increment);
```

### Argument

#### **increment**

As a positive argument, decreases priority; as a negative argument, increases priority. Issuing `nice(0)` restores the base priority. The resulting priority cannot be less than 1, or greater than the process's base priority. If it is, the `nice` function quietly does nothing.

### Description

When a process calls the `vfork` function, the resulting child inherits the parent's priority.

See also `vfork` in this section.

### Return Values

|    |                    |
|----|--------------------|
| 0  | Indicates success. |
| −1 | Indicates failure. |

## **nint** *(Alpha only)*

---

## **nint** *(Alpha only)*

Returns the nearest integral value to the argument.

### **Format**

```
#include <math.h>

double nint (double x);
float nintf (float x);
long double nintl (long double x);
```

### **Argument**

**x**  
A real number.

### **Description**

The `nint` functions return the nearest integral value to  $x$ , except halfway cases are rounded to the integral value larger in magnitude. This corresponds to the Fortran generic intrinsic function `nint`.

### **Return Values**

|                  |                                                              |
|------------------|--------------------------------------------------------------|
| <code>n</code>   | The nearest integral value to $x$ .                          |
| <code>NaN</code> | $x$ is NaN; <code>errno</code> is set to <code>EDOM</code> . |

---

**[no]nl**

The nl and nonl functions are provided only for UNIX software compatibility and have no function in the OpenVMS environment.

**Format**

```
#include <curses.h>

void nl (void);
void nonl (void);
```

---

## nl\_langinfo

Returns a pointer to a string that contains information obtained from the program's current locale.

### Format

```
#include <langinfo.h>

char *nl_langinfo (nl_item item);
```

### Argument

#### item

The name of a constant that specifies the information required. These constants are defined in `<langinfo.h>`.

The following constants are valid:

| Constant   | Category | Description                                           |
|------------|----------|-------------------------------------------------------|
| D_T_FMT    | LC_TIME  | String for formatting date and time                   |
| D_FMT      | LC_TIME  | String for formatting date                            |
| T_FMT      | LC_TIME  | String for formatting time                            |
| T_FMT_AMPM | LC_TIME  | Time format with AM/PM string                         |
| AM_STR     | LC_TIME  | String that represents AM in 12-hour clock notation   |
| PM_STR     | LC_TIME  | String that represents PM in 12-hour clock notation   |
| DAY_1      | LC_TIME  | The name of the first day of the week                 |
| ...        |          |                                                       |
| DAY_7      | LC_TIME  | The name of the seventh day of the week               |
| ABDAY_1    | LC_TIME  | The abbreviated name of the first day of the week     |
| ...        |          |                                                       |
| ABDAY_7    | LC_TIME  | The abbreviated name of the seventh day of the week   |
| MON_1      | LC_TIME  | The name of the first month in the year               |
| ...        |          |                                                       |
| MON_12     | LC_TIME  | The name of the twelfth month in the year             |
| ABMON_1    | LC_TIME  | The abbreviated name of the first month in the year   |
| ...        |          |                                                       |
| ABMON_12   | LC_TIME  | The abbreviated name of the twelfth month in the year |
| ERA        | LC_TIME  | Era description strings                               |

| Constant    | Category    | Description                                                                                                                                                                                                                                                                                                  |
|-------------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ERA_D_FMT   | LC_TIME     | Era date format string                                                                                                                                                                                                                                                                                       |
| ERA_T_FMT   | LC_TIME     | Era time format                                                                                                                                                                                                                                                                                              |
| ERA_D_T_FMT | LC_TIME     | Era date and time format                                                                                                                                                                                                                                                                                     |
| ALT_DIGITS  | LC_TIME     | Alternative symbols for digits                                                                                                                                                                                                                                                                               |
| RADIXCHAR   | LC_NUMERIC  | The radix character                                                                                                                                                                                                                                                                                          |
| THOUSEP     | LC_NUMERIC  | The character used to separate groups of digits in nonmonetary values                                                                                                                                                                                                                                        |
| YESEXp      | LC_MESSAGES | The expression for affirmative responses to yes/no questions                                                                                                                                                                                                                                                 |
| NOEXP       | LC_MESSAGES | The expression for negative responses to yes/no questions                                                                                                                                                                                                                                                    |
| CRNCYSTR    | LC_MONETARY | The currency symbol. It is preceded by one of the following: <ul style="list-style-type: none"> <li>• A minus (–) if the symbol is to appear before the value</li> <li>• A plus (+) if the symbol is to appear after the value</li> <li>• A period (.) if the symbol replaces the radix character</li> </ul> |
| CODESET     | LC_CTYPE    | Codeset name                                                                                                                                                                                                                                                                                                 |

## Description

If the current locale does not have language information defined, the function returns information from the C locale. The program should not modify the string returned by the function. This string might be overwritten by subsequent calls to `nl_langinfo`.

If the `setlocale` function is called after a call to `nl_langinfo`, then the pointer returned by the previous call to `nl_langinfo` will be unspecified. In this case, the `nl_langinfo` function should be called again.

## Return Value

**x**                      Pointer to the string containing the requested information. If *item* is invalid, the function returns an empty string.

## Example

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

/* This test sets up the British English locale, and then
/* inquires on the data and time format, first day of the week,
/* and abbreviated first day of the week. */

#include <stdlib.h>
#include <string.h>
```

## nl\_langinfo

```
int main()
{
 char *return_val;
 char *nl_ptr;

 /* set the locale, with user supplied locale name */
 return_val = setlocale(LC_ALL, "en_gb.iso8859-1");
 if (return_val == NULL) {
 printf("ERROR : The locale is unknown");
 exit(1);
 }
 printf("+-----+\n");

 /* Get the date and time format from the locale. */
 printf("D_T_FMT = ");

 /* Compare the returned string from nl_langinfo with */
 /* an empty string. */
 if (!strcmp((nl_ptr = (char *) nl_langinfo(D_T_FMT)), "")) {
 /* The string returned was empty this could mean that either */
 /* 1) The locale does not contain a value for this item */
 /* 2) The value for this item is an empty string */
 printf("nl_langinfo returned an empty string\n");
 }
 else {
 /* Display the date and time format */
 printf("%s\n", nl_ptr);
 }

 /* Get the full name for the first day of the week from locale */
 printf("DAY_1 = ");

 /* Compare the returned string from nl_langinfo with */
 /* an empty string. */
 if (!strcmp((nl_ptr = (char *) nl_langinfo(DAY_1)), "")) {
 /* The string returned was empty this could mean that either */
 /* 1) The locale does not contain a value for the first */
 /* day of the week */
 /* 2) The value for the first day of the week is */
 /* an empty string */
 printf("nl_langinfo returned an empty string\n");
 }
 else {
 /* Display the full name of the first day of the week */
 printf("%s\n", nl_ptr);
 }

 /* Get the abbreviated name for the first day of the week */
 /* from locale */
 printf("ABDAY_1 = ");

 /* Compare the returned string from nl_langinfo with an empty */
 /* string. */
 if (!strcmp((nl_ptr = (char *) nl_langinfo(ABDAY_1)), "")) {
 /* The string returned was empty this could mean that either */
 /* 1) The locale does not contain a value for the first */
 /* day of the week */
 /* 2) The value for the first day of the week is an */
 /* empty string */
 }
}
```



```

 printf("nl_langinfo returned an empty string\n");
 }
 else {
 /* Display the abbreviated name of the first day of the week */
 printf("%s\n", nl_ptr);
 }
}

```

Running the example program produces the following result:

```

+-----+
D_T_FMT = %a %e %b %H:%M:%S %Y
DAY_1 = Sunday
ABDAY_1 = Sun

```

---

**nrand48**

Generates uniformly distributed pseudorandom-number sequences. Returns 48-bit signed long integers.

**Format**

```
#include <stdlib.h>

long int nrand48 (unsigned short int xsubi[3]);
```

**Arguments*****xsubi***

An array of three short int that, when concatenated together, form a 48-bit integer.

**Description**

The nrand48 function generates pseudorandom numbers using the linear congruential algorithm and 48-bit integer arithmetic.

The nrand48 function returns nonnegative, long integers uniformly distributed over the range of  $y$  values, such that  $0 \leq y < 2^{31}$ .

The function works by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

The argument  $m$  equals  $2^{48}$ , so 48-bit integer arithmetic is performed. Unless you invoke the lcong48 function, the multiplier value  $a$  and the addend value  $c$  are:

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

The nrand48 function requires that the calling program pass an array as the *xsubi* argument, which for the first call must be initialized to the initial value of the pseudorandom-number sequence. Unlike the drand48 function, it is not necessary to call an initialization function prior to the first call.

By using different arguments, the nrand48 function allows separate modules of a large program to generate several independent sequences of pseudorandom numbers. For example, the sequence of numbers that one module generates does not depend upon how many times the functions are called by other modules.

**Return Value**

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <b><i>n</i></b> | Returns nonnegative, long integers over the range $0 \leq y < 2^{31}$ . |
|-----------------|-------------------------------------------------------------------------|

---

## open

Opens a file for reading, writing, or editing. It positions the file at its beginning (byte 0).

### Format

```
#include <fcntl.h>
```

```
int open (const char *file_spec, int flags, mode_t mode); (ANSI C)
```

```
int open (const char *file_spec, int flags, . . .); (COMPAQ C Extension)
```

### Arguments

#### file\_spec

A null-terminated character string containing a valid file specification. If you specify a directory in the *file\_spec* and it is a search list that contains an error, *Compaq C* interprets it as a file open error.

#### flags

The following values are defined in the <file.h> header file:

|          |                                             |
|----------|---------------------------------------------|
| O_RDONLY | Open for reading only                       |
| O_WRONLY | Open for writing only                       |
| O_RDWR   | Open for reading and writing                |
| O_NDELAY | Open for asynchronous input                 |
| O_APPEND | Append on each write                        |
| O_CREAT  | Create a file if it does not exist          |
| O_TRUNC  | Create a new version of this file           |
| O_EXCL   | Error if attempting to create existing file |

These flags are set using the bitwise OR operator ( | ) to separate specified flags.

Opening a file with O\_APPEND causes each write on the file to be appended to the end. (In contrast, with the VAX C RTL the behavior of files opened in append mode was to start at EOF and, thereafter, write at the current file position.)

If O\_TRUNC is specified and the file exists, open creates a new file by incrementing the version number by 1, leaving the old version in existence.

If O\_CREAT is set and the named file does not exist, the *Compaq C* RTL creates it with any attributes specified in the fourth and subsequent arguments ( . . . ). If O\_EXCL is set with O\_CREAT and the named file exists, the attempted open returns an error.

#### mode

An unsigned value that specifies the file-protection mode. The compiler performs a bitwise AND operation on the mode and the complement of the current protection mode.

You can construct modes by using the bitwise OR operator ( | ) to separate specified modes. The modes are:

```
0400 OWNER:READ
```

## open

|      |               |
|------|---------------|
| 0200 | OWNER:WRITE   |
| 0100 | OWNER:EXECUTE |
| 0040 | GROUP:READ    |
| 0020 | GROUP:WRITE   |
| 0010 | GROUP:EXECUTE |
| 0004 | WORLD:READ    |
| 0002 | WORLD:WRITE   |
| 0001 | WORLD:EXECUTE |

The system is given the same access privileges as the owner. A WRITE privilege also implies a DELETE privilege.

...

Optional file attribute arguments. The file attribute arguments are the same as those used in the `creat` function. For more information, see the `creat` function.

## Description

If a version of the file exists, a new file created with `open` inherits certain attributes from the existing file unless those attributes are specified in the `open` call. The following attributes are inherited: record format, maximum record size, carriage control, and file protection.

---

### Notes

---

- If you intend to do random writing to a file, the file must be opened for update by specifying a *flags* value of `O_RDWR`.
- To create files with OpenVMS RMS default protections by using the UNIX system-call functions `umask`, `mkdir`, `creat`, and `open`, call `mkdir`, `creat`, and `open` with a file-protection mode argument of `0777` in a program that never specifically calls `umask`. These default protections include correctly establishing protections based on ACLs, previous versions of files, and so on.

In programs that do `vfork/exec` calls, the new process image inherits whether `umask` has ever been called or not from the calling process image. The `umask` setting and whether the `umask` function has ever been called are both inherited attributes.

---

See also `creat`, `read`, `write`, `close`, `dup`, `dup2`, and `lseek` in this section.

## Return Values

|    |                                                                                                                                          |
|----|------------------------------------------------------------------------------------------------------------------------------------------|
| x  | A nonnegative file descriptor number.                                                                                                    |
| -1 | Indicates that the file does not exist, that it is protected against reading or writing, or that it cannot be opened for another reason. |

**Example**

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

main()
{
 int file,
 stat;
 int flags;

 flags = O_RDWR; /* Open for read and write, */
 /* with user default file protection, */
 /* with max fixed record size of 2048, */
 /* and a block size of 2048 bytes. */

 file = open("file.dat", flags, 0, "rfm=fix", "mrs=2048", "bls=2048");
 if (file == -1)
 perror("OPEN error"), exit(1);

 close(file);
}
```

## opendir

---

### opendir

Opens a specified directory.

#### Format

```
#include <dirent.h>

DIR *opendir (const char *dir_name);
```

#### Argument

**dir\_name**  
The name of the directory to be opened.

#### Description

The `opendir` function opens the directory specified by *dir\_name* and associates a directory stream with it. The directory stream is positioned at the first entry. The type `DIR`, defined in the `<dirent.h>` header file, represents a directory stream. A directory stream is an ordered sequence of all the directory entries in a particular directory.

The `opendir` function also returns a pointer to identify the directory stream in subsequent operations. The `NULL` pointer is returned when the directory named by *dir\_name* cannot be accessed, or when not enough memory is available to hold the entire stream.

---

#### Note

---

An open directory must always be closed with the `closedir` function to ensure that the next attempt to open that directory is successful. The `opendir` function should be used with `readdir`, `closedir`, and `rewinddir` to examine the contents of the directory.

---

#### Example

See the program example in the description of `closedir`.

#### Return Values

|   |                                                   |
|---|---------------------------------------------------|
| x | A pointer to an object of type <code>DIR</code> . |
|---|---------------------------------------------------|

NULL

Indicates an error; `errno` is set to one of the following values:

- `EACCES` – Search permission is denied for any component of *dir\_name* or read permission is denied for *dir\_name*.
- `ENAMETOOLONG` – The length of the *dir\_name* string exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX`.
- `ENOENT` – The *dir\_name* argument points to the name of a file that does not exist, or is an empty string.

## overlay

---

## overlay

Nondestructively superimposes *win1* on *win2*. The function writes the contents of *win1* that will fit onto *win2* beginning at the starting coordinates of both windows. Blanks on *win1* leave the contents of the corresponding space on *win2* unaltered. The overlay function copies as much of a window's box as possible.

### Format

```
#include <curses.h>

int overlay (WINDOW *win1, WINDOW *win2);
```

### Arguments

**win1**  
A pointer to the window.

**win2**  
A pointer to the window.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |



---

## overwrite

Destructively writes the contents of *win1* on *win2*.

### Format

```
#include <curses.h>

int overwrite (WINDOW *win1, WINDOW *win2);
```

### Arguments

**win1**  
A pointer to the window.

**win2**  
A pointer to the window.

### Description

The `overwrite` function writes the contents of *win1* that will fit onto *win2* beginning at the starting coordinates of both windows. Blanks on *win1* are written on *win2* as blanks. This function copies as much of a window's box as possible.

### Return Values

|     |                    |
|-----|--------------------|
| OK  | Indicates success. |
| ERR | Indicates failure. |

---

## pathconf

Retrieves file implementation characteristics.

### Format

```
#include <unistd.h>

long int pathconf (const char *path, int name);
```

### Arguments

#### **path**

The pathname of a file or directory.

#### **name**

The configuration attribute to query. If this attribute is not applicable to the file specified by the *path* argument, the pathconf function returns an error.

### Description

The pathconf function allows an application to determine the characteristics of operations supported by the file system underlying the file named by *path*. Read, write, or execute permission of the named file is not required, but you must be able to search all directories in the path leading to the file.

Symbolic values for the *name* argument are defined in the <unistd.h> header file, as follows:

|                            |                                                                                                                                                                                                                                                                                                    |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_PC_LINK_MAX</code>  | The maximum number of links to the file. If the <i>path</i> argument refers to a directory, the value returned applies to the directory itself.                                                                                                                                                    |
| <code>_PC_MAX_CANON</code> | The maximum number of bytes in a canonical input line. This is applicable only to terminal devices.                                                                                                                                                                                                |
| <code>_PC_MAX_INPUT</code> | The number of types allowed in an input queue. This is applicable only to terminal devices.                                                                                                                                                                                                        |
| <code>_PC_NAME_MAX</code>  | Maximum number of bytes in a file name (not including a terminating null). The byte range value is between 13 and 255. This is applicable only to a directory file. The value returned applies to file names within the directory.                                                                 |
| <code>_PC_PATH_MAX</code>  | Maximum number of bytes in a path name (not including a terminating null). The value is never larger than 65,535. This is applicable only to a directory file. The value returned is the maximum length of a relative pathname when the specified directory is the working directory.              |
| <code>_PC_PIPE_BUF</code>  | Maximum number of bytes guaranteed to be written atomically. This is applicable only to a FIFO. The value returned applies to the referenced object. If the <i>path</i> argument refers to a directory, the value returned applies to any FIFO that exists or can be created within the directory. |

|                                   |                                                                                                                                                                                                            |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>_PC_CHOWN_RESTRICTED</code> | This is applicable only to a directory file. The value returned applies to any files (other than directories) that exist or can be created within the directory.                                           |
| <code>_PC_NO_TRUNC</code>         | Returns 1 if supplying a component name longer than allowed by <code>NAME_MAX</code> causes an error. Returns 0 (zero) if long component names are truncated. This is applicable only to a directory file. |
| <code>_PC_VDISABLE</code>         | This is always 0 (zero); no disabling character is defined. This is applicable only to a terminal device.                                                                                                  |

## Return Values

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>  | Resultant value of the configuration attribute specified in <i>name</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <code>-1</code> | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"> <li>• <code>EACCES</code> – Search permission is denied for a component of the path prefix.</li> <li>• <code>EINVAL</code> – The <i>name</i> argument specifies an unknown or inapplicable characteristic.</li> <li>• <code>EFAULT</code> – The <i>path</i> argument is an invalid address.</li> <li>• <code>ENAMETOOLONG</code> – The length of the <i>path</i> string exceeds <code>PATH_MAX</code> or a pathname component is longer than <code>NAME_MAX</code>.</li> <li>• <code>ENOENT</code> – The named file does not exist or the <i>path</i> argument points to an empty string.</li> <li>• <code>ENOTDI</code> – A component of the <i>path</i> prefix is not a directory.</li> </ul> |

## pause

---

## pause

Suspends the calling process until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

### Format

```
#include <unistd.h>

int pause (void);
```

### Description

The pause function suspends the calling process until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

If the action is to terminate the process, pause does not return.

If the action is to execute a signal-catching function, pause returns after the signal-catching function returns.

### Return Value

|    |                                                                                                                                                  |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------|
|    | Since the pause function suspends process execution indefinitely unless interrupted by a signal, there is no successful completion return value. |
| −1 | In cases where pause returns, the return value is −1, and errno is set to EINTR.                                                                 |

---

## pclose

Closes a pipe to a process.

### Format

```
#include <stdio.h>

int pclose (FILE *stream);
```

### Arguments

**stream**

A pointer to a FILE structure for an open pipe returned by a previous call to the popen function.

### Description

The pclose function closes a pipe between the calling program and a shell command to be executed. Use pclose to close any stream you have opened with popen. The pclose function waits for the associated process to end, and then returns the exit status of the command. See the description of waitpid for information on interpreting the exit status.

Beginning with OpenVMS Version 7.3-1, when compiled with the \_VMS\_WAIT macro defined, the pclose function returns the OpenVMS completion code of the child process.

See also popen in this section.

### Return Values

|    |                                                                                                                                                                                                                                         |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | Exit status of child.                                                                                                                                                                                                                   |
| −1 | Indicates an error. The <i>stream</i> argument is not associated with a popen function. errno is set to one of the following: <ul style="list-style-type: none"><li>• ECHILD – cannot obtain the status of the child process.</li></ul> |

## perror

---

## perror

Writes a short error message to stderr describing the current value of errno.

### Format

```
#include <stdio.h>

void perror (const char *str);
```

### Argument

#### **str**

Usually the name of the program that caused the error.

### Description

The perror function uses the error number in the external variable errno to retrieve the appropriate locale-dependent error message. The function writes out the message as follows: *str* (a user-supplied prefix to the error message), followed by a colon and a space, followed by the message itself, followed by a new-line character.

See the description of errno in Chapter 4 for a list of possible errors.

See also strerror in this section.

### Example

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

main(argc, argv)
 int argc;
 char *argv[];
{
 FILE *fp;

 fp = fopen(argv[1], "r"); /* Open an input file. */
 if (fp == NULL) {
 /* If the fopen call failed, perror prints out a */
 /* diagnostic: */
 /* "open: <error message>" */
 /* This error message provides a diagnostic explaining */
 /* the cause of the failure. */
 perror("open");
 exit(EXIT_FAILURE);
 }
 else
 fclose(fd);
}
```

---

## pipe

Creates a temporary mailbox that can be used to read and write data between a parent and child process. The channels through which the processes communicate are called a *pipe*.

### Format

```
#include <unistd.h>

int pipe (int array_fdscptr[2]); (ISO POSIX-1)

int pipe (int array_fdscptr[2], ...); (COMPAQ C Extension)
```

### Arguments

#### array\_fdscptr

An array of file descriptors. A pipe is implemented as an array of file descriptors associated with a mailbox. These mailbox descriptors are special in that these are the only file descriptors which, when passed to the `isapipe` function, will return 1.

The file descriptors are allocated in the following way:

- The first available file descriptor is assigned to writing, and the next available file descriptor is assigned to reading.
- The file descriptors are then placed in the array in reverse order; element 0 contains the file descriptor for reading, and element 1 contains the file descriptor for writing.

...

Represents two optional, positional arguments, *flag* and *bufsize*, which follow:

#### flag

An optional argument used as a bitmask.

If either the `O_NDELAY` or `O_NONBLOCK` bit is set, the I/O operations to the mailbox through *array\_fdscptr* file descriptors terminate immediately, rather than waiting for another process.

If, for example, the `O_NDELAY` bit is set and the child issues a read request to the mailbox before the parent has put any data into it, the read terminates immediately with zero status. If neither the `O_NDELAY` nor `O_NONBLOCK` bit is set, the child will be waiting on the read until the parent writes any data into the mailbox. This is the default behavior if no *flag* argument is specified.

The values of `O_NDELAY` and `O_NONBLOCK` are defined in the `<fcntl.h>` header file. Any other bits in the *flag* argument are ignored. You must specify this argument if the second optional, positional argument *bufsize* is specified. If the *flag* argument is needed only to allow specification of the *bufsize* argument, specify *flag* as zero.

#### bufsize

Optional argument of type `int` that specifies the size of the mailbox, in bytes. If you do not specify this argument, *Compaq C* for *OpenVMS* Systems creates a mailbox with a default size of 512 bytes.

If you do specify this argument, be sure to precede it with a *flag* argument of 0.

## pipe

### Description

The mailbox used for the pipe is a temporary mailbox. The mailbox is not deleted until all processes that have open channels to that mailbox close those channels. The last process that closes a pipe writes a message to the mailbox, indicating the end-of-file.

The mailbox is created by using the \$CREMBX system service, specifying the following characteristics:

- A maximum message length of 512 characters
- A buffer quota of 512 characters
- A protection mask granting all privileges to USER and GROUP and no privileges to SYSTEM or WORLD

The buffer quota of 512 characters implies that you cannot write more than 512 characters to the mailbox before all or part of the mailbox is read. Since a mailbox record is slightly larger than the data part of the message that it contains, not all of the 512 characters can be used for message data. You can increase the size of the buffer by specifying an alternative size using the optional, third argument to the pipe function. A pipe under the OpenVMS system is a stream-oriented file with no carriage-control attributes. It is fully buffered by default in the *Compaq C* RTL. A mailbox used as a pipe is different than a mailbox created by the application. A mailbox created by the application defaults to a record-oriented file with carriage return, carriage control. Additionally, writing a zero-length record to a mailbox writes an EOF, as does each close of the mailbox. For a pipe, only the last close of a pipe writes an EOF.

The pipe is created by the parent process before vfork and an exec function are called. By calling pipe first, the child inherits the open file descriptors for the pipe. You can then use the getname function to return the name of the mailbox associated with the pipe, if this information is desired. The mailbox name returned by getname has the format `_MBAnnnn:`, where `nnnn` is a unique number.

Both the parent and the child need to know in advance which file descriptors will be allocated for the pipe. This information cannot be retrieved at run time. Therefore, it is important to understand how file descriptors are used in any *Compaq C* for *OpenVMS* program. For more information about file descriptors, see Chapter 2.

File descriptors 0, 1, and 2 are open in a *Compaq C* for *OpenVMS* program for stdin (SYS\$INPUT), stdout (SYS\$OUTPUT), and stderr (SYS\$ERROR), respectively. Therefore, if no other files are open when pipe is called, pipe assigns file descriptor 3 for writing and file descriptor 4 for reading. In the array returned by pipe, 4 is placed in element 0 and 3 is placed in element 1.

If other files have been opened, pipe assigns the first available file descriptor for writing and the next available file descriptor for reading. In this case, the pipe does not necessarily use adjacent file descriptors. For example, assume that two files have been opened and assigned to file descriptors 3 and 4 and the first file is then closed. If pipe is called at this point, file descriptor 3 is assigned for writing and file descriptor 5 is assigned for reading. Element 0 of the array will contain 5 and element 1 will contain 3.



In large applications that do large amounts of I/O, it gets more difficult to predict which file descriptors are going to be assigned to a pipe; and, unless the child knows which file descriptors are being used, it will not be able to read and write successfully from and to the pipe.

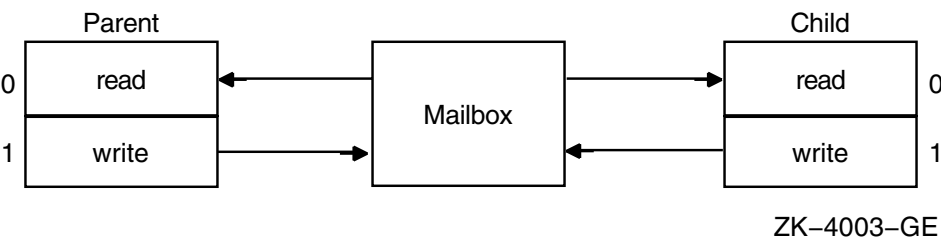
One way to be sure that the correct file descriptors are being used is to use the following procedure:

- 1. Choose two descriptor numbers that will be known to both the parent and the child. The numbers should be high enough to account for any I/O that might be done before the pipe is created.
- 2. Call pipe in the parent at some point before calling an exec function.
- 3. In the parent, use dup2 to assign the file descriptors returned by pipe to the file descriptors you chose. This now reserves those file descriptors for the pipe; any subsequent I/O will not interfere with the pipe.

You can read and write through the pipe using the UNIX I/O functions read and write, specifying the appropriate file descriptors. As an alternative, you can issue fdopen calls to associate file pointers with these file descriptors so that you can use the Standard I/O functions (fread and fwrite).

Two separate file descriptors are used for reading from and writing to the pipe, but only one mailbox is used so some I/O synchronization is required. For example, assume that the parent writes a message to the pipe. If the parent is the first process to read from the pipe, then it will read its own message back as shown in Figure REF-1.

Figure REF-1 Reading and Writing to a Pipe



Return Values

- 0 Indicates success.
- 1 Indicates an error.

## popen

---

## popen

Initiates a pipe to a process.

### Format

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

```
FILE *popen (const char *command, const char *type);
```

### Arguments

#### **command**

A pointer to a null-terminated string containing a shell command line.

#### **type**

A pointer to a null-terminated string containing an I/O mode. Because open files are shared, you can use a type `r` command as an input filter and a type `w` command as an output filter. Specify one of the following values for the *type* argument:

- `r` – the calling program can read from the standard output of the command by reading from the returned file stream.
- `w` – the calling program can write to the standard input of the command by writing to the returned file stream.

### Description

The `popen` function creates a pipe between the calling program and a shell command awaiting execution. It returns a pointer to a `FILE` structure for the stream.

---

#### **Note**

When you use the `popen` function to invoke an output filter, beware of possible deadlock caused by output data remaining in the program buffer. You can avoid this by either using the `setvbuf` function to ensure that the output stream is unbuffered, or the `fflush` function to ensure that all buffered data is flushed before calling the `pclose` function.

---

See also `fflush`, `pclose`, and `setvbuf` in this section.

### Return Values

|                   |                                                                     |
|-------------------|---------------------------------------------------------------------|
| <code>x</code>    | A pointer to the <code>FILE</code> structure for the opened stream. |
| <code>NULL</code> | Indicates an error. Unable to create files or processes.            |

---

## pow

Returns the first argument raised to the power of the second argument.

### Format

```
#include <math.h>

double pow (double x, double y);
float powf (float x, float y); (Alpha only)
long double powl (long double x, long double y); (Alpha only)
```

### Arguments

**x**  
A floating-point base to be raised to an exponent *y*.

**y**  
The exponent to which the base *x* is to be raised.

### Description

The `pow` functions raise a floating-point base *x* to a floating-point exponent *y*. The value of `pow(x,y)` is computed as  $e^{*(y \ln(x))}$  for positive *x*.

If *x* is 0 and *y* is negative, `−HUGE_VAL` is returned, and `errno` is set to `EDOM`.

### Return Values

|                        |                                                                                              |
|------------------------|----------------------------------------------------------------------------------------------|
| <code>x</code>         | The result of the first argument raised to the power of the second.                          |
| <code>1.0</code>       | The base is 0 and the exponent is 0.                                                         |
| <code>HUGE_VAL</code>  | The result overflowed; <code>errno</code> is set to <code>ERANGE</code> .                    |
| <code>−HUGE_VAL</code> | The base is 0 and the exponent is negative; <code>errno</code> is set to <code>EDOM</code> . |

### Example

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

main()
{
 double x;
 errno = 0;
 x = pow(-3.0, 2.0);
 printf("%d, %f\n", errno, x);
}
```

This example program outputs the following:

```
0, 9.000000
```

## printf

---

## printf

Performs formatted output from the standard output (stdout). See Chapter 2 for information on format specifiers.

### Format

```
#include <stdio.h>

int printf (const char *format_spec, ...);
```

### Arguments

#### **format\_spec**

Characters to be written literally to the output or converted as specified in the ... arguments.

...

Optional expressions whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, you may omit the output sources. Otherwise, the function call must have exactly as many output sources as there are conversion specifications, and the conversion specifications must match the types of the output sources.

Conversion specifications are matched to output sources in left-to-right order. Excess output pointers, if any, are ignored.

### Return Values

|                |                                                                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| x              | The number of bytes written.                                                                                                                    |
| Negative value | Indicates that an output error occurred. The function sets errno. For a list of errno values set by this function, see fprintf in this section. |

---

## **[w]printw**

Perform a `printf` in the specified window, starting at the current position of the cursor. The `printw` function acts on the `stdscr` window.

### **Format**

```
#include <curses.h>

printw (char *format_spec, ...);
int wprintw (WINDOW *win, char *format_spec, ...);
```

### **Arguments**

#### **win**

A pointer to the window.

#### **format\_spec**

A pointer to the format specification string.

...

Optional expressions whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, you may omit the output sources. Otherwise, the function call must have exactly as many output sources as there are conversion specifications, and the conversion specifications must match the types of the output sources.

Conversion specifications are matched to output sources in left-to-right order. Excess output pointers, if any, are ignored.

### **Description**

The formatting specification (*format\_spec*) and the other arguments are identical to those used with the `printf` function.

The `printw` and `wprintw` functions format and then print the resultant string to the window using the `addstr` function. For more information, see the `printf` and `scrollok` functions in this section. See Chapter 2 for information on format specifiers.

### **Return Values**

|     |                                                                |
|-----|----------------------------------------------------------------|
| OK  | Indicates success.                                             |
| ERR | Indicates that the function makes the window scroll illegally. |

## putc

---

## putc

The `putc` macro writes a single character to a specified file.

### Format

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

```
int putc (int character, FILE *file_ptr);
```

### Arguments

**character**

An object of type `int`.

**file\_ptr**

A file pointer.

### Description

Since `putc` is a macro, a file pointer argument with side effects (for example, `putc (ch, *f++)`) might be evaluated incorrectly. In such a case, use the `fputc` function instead. See also `fputc` in this section.

### Return Values

|                  |                                                       |
|------------------|-------------------------------------------------------|
| <code>x</code>   | The character written to the file. Indicates success. |
| <code>EOF</code> | Indicates output errors.                              |

---

## putchar

Writes a single character to the standard output (stdout) and returns the character.

### Format

```
#include <stdio.h>

int putchar (int character);
```

### Argument

**character**  
An object of type int.

### Description

The putchar function is identical to fputc (*character*, stdout).

### Return Values

|           |                          |
|-----------|--------------------------|
| character | Indicates success.       |
| EOF       | Indicates output errors. |

## putenv

---

## putenv

Sets an environmental variable.

### Format

```
#include <stdlib.h>

int putenv (const char *string);
```

### Argument

#### **string**

A pointer to a *name=value* string.

### Description

The putenv function sets the value of an environment variable by altering an existing variable or by creating a new one. The *string* argument points to a string of the form *name=value*, where *name* is the environment variable and *value* is the new value for it.

The string pointed to by *string* becomes part of the environment, so altering the string changes the environment. When a new string-defining name is passed to putenv, the space used by *string* is no longer used.

---

#### **Note**

---

The putenv function manipulates the environment pointed to by the environ external variable, and can be used with getenv. However, the third argument to the main function (the environment pointer), is not changed.

The putenv function uses the malloc function to enlarge the environment.

A potential error is to call putenv with an automatic variable as the argument, then exit the calling function while *string* is still part of the environment.

---

### Return Values

|     |                                                                                                               |
|-----|---------------------------------------------------------------------------------------------------------------|
| 0   | Indicates success.                                                                                            |
| − 1 | Indicates an error. errno is set to ENOMEM—<br>Not enough memory available to expand the<br>environment list. |

### Restriction

The putenv function cannot take a 64-bit address. See Section 1.10.



---

## puts

Writes a character string to the standard output (stdout) followed by a new-line character.

### Format

```
#include <stdio.h>

int puts (const char *str);
```

### Argument

**str**  
A pointer to a character string.

### Description

The puts function does not copy the terminating null character to the output stream.

### Return Values

|                   |                          |
|-------------------|--------------------------|
| Nonnegative value | Indicates success.       |
| EOF               | Indicates output errors. |

## putw

---

## putw

Writes characters to a specified file.

### Format

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

```
int putw (int integer, FILE *file_ptr);
```

### Arguments

**integer**

An object of type int or long.

**file\_ptr**

A file pointer.

### Description

The putw function writes four characters to the output file as an int. No conversion is performed.

### Return Values

integer

Indicates success.

EOF

Indicates output errors.

---

## putwc

Converts a wide character to its corresponding multibyte value, and writes the result to a specified file.

### Format

```
#include <wchar.h>

wint_t putwc (wint_t wc, FILE *file_ptr);
```

### Arguments

**wc**  
An object of type `wint_t`.

**file\_ptr**  
A file pointer.

### Description

Since `putwc` might be implemented as a macro, a file pointer argument with side effects (for example `putwc (wc, *f++)`) might be evaluated incorrectly. In such a case, use the `fputwc` function instead.

See also `fputwc` in this section.

### Return Values

|                   |                                                                                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>    | The character written to the file. Indicates success.                                                                                                                        |
| <code>WEOF</code> | Indicates an output error. The function sets <code>errno</code> . For a list of the <code>errno</code> values set by this function, see <code>fputwc</code> in this section. |

## putwchar

---

## putwchar

Writes a wide character to the standard output (stdout) and returns the character.

### Format

```
#include <wchar.h>

wint_t putwchar (wint_t wc);
```

### Arguments

**wc**  
An object of type `wint_t`.

### Description

The `putwchar` function is identical to `fputwc(wc, stdout)`.

### Return Values

|                   |                                                                                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>    | The character written to the file. Indicates success.                                                                                                                        |
| <code>WEOF</code> | Indicates an output error. The function sets <code>errno</code> . For a list of the <code>errno</code> values set by this function, see <code>fputwc</code> in this section. |

---

**qabs, llabs** *(Alpha only)*

Returns the absolute value of an integer as an `__int64`. `llabs` is a synonym for `qabs`.

**Format**

```
#include <stdlib.h>

__int64 qabs (__int64 j);
__int64 llabs (__int64 j);
```

**Argument**

**j**  
A value of type `__int64`.

## **qdiv, lldiv** *(Alpha only)*

---

## **qdiv, lldiv** *(Alpha only)*

Returns the quotient and the remainder after the division of its arguments.  
lldiv is a synonym for qdiv.

### **Format**

```
#include <stdlib.h>

qdiv_t qdiv (__int64 numer, __int64 denom);
lldiv_t lldiv (__int64 numer, __int64 denom);
```

### **Arguments**

**numer**

A numerator of type `__int64`.

**denom**

A denominator of type `__int64`.

### **Description**

The types `qdiv_t` and `lldiv_t` are defined in the `<stdlib.h>` header file as follows:

```
typedef struct
{
 __int64 quot, rem;
} qdiv_t, lldiv_t;
```

---

## qsort

Sorts an array of objects in place. It implements the quick-sort algorithm.

### Format

```
#include <stdlib.h>

void qsort (void *base, size_t nmemb, size_t size, int (*compar) (const void *, const void *));
```

### Function Variants

The `qsort` function has variants named `_qsort32` and `_qsort64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**base**

A pointer to the first member of the array. The pointer should be of type pointer-to-element and cast to type pointer-to-character.

**nmemb**

The number of objects in the array.

**size**

The size of an object, in bytes.

**compar**

A pointer to the comparison function.

### Description

Two arguments are passed to the comparison function pointed to by *compar*. The two arguments point to the objects being compared. Depending on whether the first argument is less than, equal to, or greater than the second argument, the comparison function returns an integer less than, equal to, or greater than 0.

The comparison function *compar* need not compare every byte, so arbitrary data might be contained in the objects in addition to the values being compared.

The order in the output of two objects that compare as equal is unpredictable.

## raise

---

## raise

Generates a specified software signal. Generating a signal causes the action routine established by the signal, `ssignal`, or `sigvec` function to be invoked.

### Format

```
#include <signal.h>

int raise (int sig); (ANSI C)

int raise (int sig, int sigcode); (COMPAQ C Extension)
```

### Arguments

#### **sig**

The signal to be generated.

#### **sigcode**

An optional signal code, available only when not compiling in strict ANSI C mode. For example, signal SIGFPE—the arithmetic trap signal—has 10 different codes, each representing a different type of arithmetic trap.

The signal codes can be represented by mnemonics or numbers. The arithmetic trap codes are represented by the numbers 1 to 10; the SIGILL codes are represented by the numbers 0 to 2. The code values are defined in the `<signal.h>` header file. See Tables 4–4 and 4–5 for a list of signal mnemonics, codes, and corresponding OpenVMS exceptions.

### Description

Calling the `raise` function has one of the following results:

- If `raise` specifies a *sig* argument that is outside the range defined in the `<signal.h>` header file, then the `raise` function returns 0, and the `errno` variable is set to `EINVAL`.
- If `signal`, `ssignal`, or `sigvec` establishes `SIG_DFL` (default action) for the signal, then the functions do not return. The image is exited with the OpenVMS error code corresponding to the signal.
- If `signal`, `ssignal`, or `sigvec` establishes `SIG_IGN` (ignore signal) as the action for the signal, then `raise` returns its argument, *sig*.
- `signal`, `ssignal`, or `sigvec` must establish an action function for the signal. That function is called and its return value is returned by `raise`.

See Chapter 4 for more information on signal processing.

See also `gsignal`, `signal`, `ssignal`, and `sigvec` in this section.

### Return Values

|         |                  |
|---------|------------------|
| 0       | If successful.   |
| nonzero | If unsuccessful. |



---

## rand

Returns pseudorandom numbers in the range 0 to  $2^{31} - 1$ .

### Format

```
#include <math.h>

int rand (void);
```

### Description

The rand function uses the following ANSI Standard algorithm to return a random number:

```
static unsigned int next = 1;
int rand(void)
{
 next = next * 1103515245 + 12345;
 return (next & RAND_MAX);
}
```

See also srand in this section.

For other random-number algorithms, see random and all the \*48 functions.

## random

---

## random

Generates pseudorandom numbers in a more random sequence.

### Format

```
#include <stdlib.h>

long int random (void);
```

### Description

The random function is a random-number generator that has virtually the same calling sequence and initialization properties as the rand function, but produces sequences that are more random. The low 12 bits generated by rand go through a cyclic pattern. All bits generated by random are usable. For example, random() & 1 produces a random binary value.

The random function uses a nonlinear, additive-feedback, random-number generator employing a default state-array size of 31 integers to return successive pseudorandom numbers in the range from 0 to  $2^{31} - 1$ . The period of this random-number generator is approximately  $16 \cdot (2^{31} - 1)$ . The size of the state array determines the period of the random-number generator. Increasing the state array size increases the period.

With a full 256 bytes of state information, the period of the random-number generator is greater than  $2^{69}$ , and is sufficient for most purposes.

Like the rand function, the random function produces by default a sequence of numbers that you can duplicate by calling the srand function with a value of 1 as the seed. The srand function, unlike the srand function, does not return the old seed because the amount of state information used is more than a single word.

See also rand, srand, srandom, setstate, and initstate in this section.

### Return Value

|   |                  |
|---|------------------|
| n | A random number. |
|---|------------------|

---

**[no]raw**

Raw mode only works with the Curses input routines `[w]getch` and `[w]getstr`. Raw mode is not supported with the *Compaq C* RTL emulation of UNIX I/O, Terminal I/O, or Standard I/O.

**Format**

```
#include <curses.h>

raw()

noraw()
```

**Description**

Raw mode reads are satisfied on one of two conditions: after a minimum number (5) of characters are input at the terminal or after waiting a fixed time (10 seconds) from receipt of any characters from the terminal.

**Example**

```
/* Example of standard and raw input in Curses package. */
#include <curses.h>
main()
{
 WINDOW *win1;
 char vert = '.',
 hor = '.',
 str[80];

 /* Initialize standard screen, turn echo off. */
 initscr();
 noecho();

 /* Define a user window. */
 win1 = newwin(22, 78, 1, 1);
 leaveok(win1, TRUE);
 leaveok(stdscr, TRUE);
 box(stdscr, vert, hor);

 /* Reset the video, refresh(redraw) both windows. */
 mvwaddstr(win1, 2, 2, "Test line terminated input");
 wrefresh(win1);

 /* Do some input and output it. */
 nocrmode();
 wgetstr(win1, str);

 mvwaddstr(win1, 5, 5, str);
 mvwaddstr(win1, 7, 7, "Type something to clear screen");
 wrefresh(win1);

 /* Get another character then delete the window. */
 wgetch(win1);
 wclear(win1);

 mvwaddstr(win1, 2, 2, "Test raw input");
 wrefresh(win1);
}
```

## [no]raw

```
/* Do some raw input 5 chars or timeout - and output it. */
raw();
getstr(str);
noraw();
mvwaddstr(win1, 5, 5, str);
mvwaddstr(win1, 7, 7, "Raw input completed");
wrefresh(win1);
endwin();
}
```

---

## read

Reads bytes from a file and places them in a buffer.

### Format

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

```
ssize_t read (int file_desc, void *buffer, size_t nbytes); (ISO POSIX-1)
```

```
int read (int file_desc, void *buffer, int nbytes); (Compatibility)
```

### Arguments

#### **file\_desc**

A file descriptor. The specified file descriptor must refer to a file currently opened for reading.

#### **buffer**

The address of contiguous storage in which the input data is placed.

#### **nbytes**

The maximum number of bytes involved in the read operation.

### Description

The read function returns the number of bytes read. The return value does not necessarily equal *nbytes*. For example, if the input is from a terminal, at most one line of characters is read.

---

#### Note

---

The read function does not span record boundaries in a record file and, therefore, reads at most one record. A separate read must be done for each record.

---

### Return Values

|    |                                                                                                                                                     |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| n  | The number of bytes read.                                                                                                                           |
| −1 | Indicates a read error, including physical input errors, illegal buffer addresses, protection violations, undefined file descriptors, and so forth. |

### Example

```
#include <file.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
```

## read

```
main()
{
 int fd,
 i;
 char buf[10];
 FILE *fp ; /* Temporary STDIO file */

 /* Create a dummy data file */
 if ((fp = fopen("test.txt", "w+")) == NULL) {
 perror("open");
 exit(1);
 }
 fputs("XYZ\n",fp) ;
 fclose(fp) ;

 /* And now practice "read" */
 if ((fd = open("test.txt", O_RDWR, 0, "shr=upd")) <= 0) {
 perror("open");
 exit(0);
 }

 /* Read 2 characters into buf. */
 if ((i = read(fd, buf, 2)) < 0) {
 perror("read");
 exit(0);
 }

 /* Print out what was read. */
 if (i > 0)
 printf("buf='%c%c'\n", buf[0], buf[1]);
 close(fd);
}
```

---

## readdir, readdir\_r

Finds entries in a directory.

### Format

```
#include <dirent.h>

struct dirent *readdir (DIR *dir_pointer);

int readdir_r (DIR *dir_pointer, struct dirent *entry, struct dirent **result);
```

### Arguments

**dir\_pointer**

A pointer to the `dir` structure of an open directory.

**entry**

A pointer to a `dirent` structure that will be initialized with the directory entry at the current position of the specified stream.

**result**

Upon successful completion, the location where a pointer to *entry* is stored.

### Description

The `readdir` function returns a pointer to a structure representing the directory entry at the current position in the directory stream specified by *dir\_pointer*, and positions the directory stream at the next entry. It returns a NULL pointer upon reaching the end of the directory stream. The `dirent` structure defined in the `<dirent.h>` header file describes a directory entry.

The type `DIR` defined in the `<dirent.h>` header file represents a directory stream. A directory stream is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files. You can remove files from or add files to a directory asynchronously to the operation of the `readdir` function.

The pointer returned by the `readdir` function points to data that you can overwrite by another call to `readdir` on the same directory stream. This data is not overwritten by another call to `readdir` on a different directory stream.

If a file is removed from or added to the directory after the most recent call to the `opendir` or `rewinddir` function, a subsequent call to the `readdir` function might not return an entry for that file.

When it reaches the end of the directory, or when it detects an invalid `seekdir` operation, the `readdir` function returns the null value.

An attempt to seek to an invalid location causes the `readdir` function to return the null value the next time it is called. A previous `telldir` function call returns the position.

The `readdir_r` function is a reentrant version of `readdir`. In addition to *dir\_pointer*, you must specify a pointer to a `dirent` structure in which the current directory entry of the specified stream is returned.

If the operation is successful, `readdir_r` returns 0 and stores one of the two following pointers in *result*:

- Pointer to *entry* if the entry was found

## readdir, readdir\_r

- NULL pointer if the end of the directory stream was reached

If an error occurred, an error value is returned that indicates the cause of the error.

The storage pointed to by *entry* must be large enough for a `dirent` with an array of `char d_name` member containing at least `NAME_MAX + 1` elements.

### Example

See the description of `closedir` for an example.

### Return Values

|      |                                                                                                                                                                   |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x    | On successful completion of <code>readdir</code> , a pointer to an object of type <code>struct dirent</code> .                                                    |
| 0    | Successful completion of <code>readdir_r</code> .                                                                                                                 |
| x    | On error, an error value ( <code>readdir_r</code> only).                                                                                                          |
| NULL | An error occurred or end of the directory stream ( <code>readdir_r</code> only). If an error occurred, <code>errno</code> is set to a value indicating the cause. |



---

## realloc

Changes the size of the area pointed to by the first argument to the number of bytes given by the second argument. These functions are AST-reentrant.

### Format

```
#include <stdlib.h>

void *realloc (void *ptr, size_t size);
```

### Function Variants

The realloc function has variants named `_realloc32` and `_realloc64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**ptr**  
Points to an allocated area, or can be NULL.

**size**  
The new size of the allocated area.

### Description

If *ptr* is the NULL pointer, the behavior of the realloc function is identical to the malloc function.

The contents of the area are unchanged up to the lesser of the old and new sizes. The ANSI C Standard states that, "If the new size is larger than the old size, the value of the newly allocated portion of memory is indeterminate." For compatibility with old implementations, *Compaq C* initializes the newly allocated memory to 0.

For efficiency, the previous actual allocation could have been larger than the requested size. If it was allocated with malloc, the value of the portion of memory between the previous requested allocation and the actual allocation is indeterminate. If it was allocated with calloc, that same memory was initialized to 0. If your application relies on realloc initializing memory to 0, then use calloc instead of malloc to perform the initial allocation.

See also free, cfree, calloc, and malloc in this section.

### Return Values

|      |                                                                                                                                                                                                                      |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x    | The address of the area, quadword-aligned. The address is returned because the area may have to be moved to a new address to reallocate enough space. If the area was moved, the space previously occupied is freed. |
| NULL | Indicates that space cannot be reallocated (for example, if there is not enough room).                                                                                                                               |

## [w]refresh

---

## [w]refresh

Repaint the specified window on the terminal screen. The refresh function acts on the `stdscr` window.

### Format

```
#include <curses.h>

int refresh();

int wrefresh (WINDOW *win);
```

### Argument

**win**

A pointer to the window.

### Description

The result of this process is that the portion of the window not occluded by subwindows or other windows appears on the terminal screen. To see the entire occluded window on the terminal screen, call the `touchwin` function instead of the `refresh` or `wrefresh` function.

See also `touchwin` in this section.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

---

## remove

Deletes a file.

### Format

```
#include <stdio.h>

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

### Argument

#### **file\_spec**

A pointer to the string that is an OpenVMS or a UNIX style file specification. The file specification can include a wildcard in its version number. So, for example, files of the form *filename.txt;\** can be deleted.

### Description

If you specify a directory in the file name and it is a search list that contains an error, *Compaq C* for *OpenVMS* Systems interprets it as a file error.

The `remove` and `delete` functions are functionally equivalent in the *Compaq C* RTL.

See also `delete` in this section.

### Return Values

|               |                    |
|---------------|--------------------|
| 0             | Indicates success. |
| nonzero value | Indicates failure. |

## rename

---

## rename

Gives a new name to an existing file.

### Format

```
#include <stdio.h>

int rename (const char *old_file_spec, const char *new_file_spec);
```

### Arguments

#### **old\_file\_spec**

A pointer to a string that is the existing name of the file to be renamed.

#### **new\_file\_spec**

A pointer to a string that is to be the new name of the file.

### Description

If you try to rename a file that is currently open, the behavior is undefined. You cannot rename a file from one physical device to another. Both the old and new file specifications must reside on the same device.

If the *new\_file\_spec* does not contain a file extension, the file extension of *old\_file\_spec* is used. To rename a file to have no file extension, *new\_file\_spec* must contain a period (.). For example, the following renames SYS\$DISK:[ ]FILE.DAT to SYS\$DISK:[ ]FILE1.DAT:

```
rename("file.dat", "file1");
```

However, the following renames SYS\$DISK:[ ]FILE.DAT to SYS\$DISK:[ ]FILE1:

```
rename(file.dat, "file1.");
```

---

#### **Note**

---

Because the rename function does special processing of the file extension, the caller must be careful when specifying the name of the renamed file in a call to a C Run-Time Library function that accepts a file-name argument. For example, after the following call to the rename function, the new file should be opened as fopen("bar.dat",...):

```
rename("foo.dat", "bar");
```

---

### Return Values

|               |                    |
|---------------|--------------------|
| 0             | Indicates success. |
| nonzero value | Indicates failure. |

---

## rewind

Sets the file to its beginning.

### Format

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

```
void rewind (FILE *file_ptr); (ISO POSIX-1)
```

```
int rewind (FILE *file_ptr); (COMPAQ C Extension)
```

### Argument

**file\_ptr**

A file pointer.

### Description

The `rewind` function is equivalent to `fseek (file_ptr, 0, SEEK_SET)`. You can use the `rewind` function with either record or stream files.

A successful call to `rewind` clears the error indicator for the file.

The ANSI C standard defines `rewind` as not returning a value; therefore, the function prototype for `rewind` is declared with a return type of `void`. However, since a `rewind` can fail, and since previous versions of the *Compaq C* RTL have declared `rewind` to return an `int`, the code for `rewind` does return 0 on success and `-1` on failure.

See also `fseek` in this section.

## rewinddir

---

## rewinddir

Resets the position of the specified directory stream to the beginning of a directory.

### Format

```
#include <dirent.h>

void rewinddir (DIR *dir_pointer);
```

### Arguments

**dir\_pointer**

A pointer to the `dir` structure of an open directory.

### Description

The `rewinddir` function resets the position of the specified directory stream to the beginning of the directory. It also causes the directory stream to refer to the current state of the corresponding directory, the same as using the `opendir` function. If the *dir\_pointer* argument does not refer to a directory stream, the effect is undefined.

The type `DIR`, defined in the `<dirent.h>` header file, represents a directory stream. A directory stream is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files.

See also `opendir` in this section.

---

## rindex

Searches for a character in a string.

### Format

```
#include <strings.h>

char *rindex (const char *s, int c);
```

### Function Variants

The `rindex` function has variants named `_rindex32` and `_rindex64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**s**  
The string to search.

**c**  
The character to search for.

### Description

The `rindex` function is identical to the `strchr` function, and is provided for compatibility with some UNIX implementations.

## **rint** *(Alpha only)*

---

## **rint** *(Alpha only)*

Rounds its argument to an integral value according to the current IEEE rounding direction specified by the user.

### **Format**

```
#include <math.h>

double rint (double x);
float rintf (float x);
long double rintl (long double x);
```

### **Argument**

**x**  
A real number.

### **Description**

The rint functions return the nearest integral value to  $x$  in the direction of the current IEEE rounding mode specified on the `/ROUNDING_MODE` command-line qualifier.

If the current rounding mode rounds toward negative Infinity, then rint is identical to floor. If the current rounding mode rounds toward positive Infinity, then rint is identical to ceil.

If  $|x| = \text{Infinity}$ , rint returns  $x$ .

### **Return Values**

|     |                                                                                       |
|-----|---------------------------------------------------------------------------------------|
| n   | The nearest integral value to $x$ in the direction of the current IEEE rounding mode. |
| NaN | $x$ is NaN; errno is set to EDOM.                                                     |



---

## rmdir

Removes a directory file.

### Format

```
#include <unistd.h>

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

### Arguments

**path**  
A directory pathname.

### Description

The `rmdir` function removes a directory file whose name is specified in the *path* argument. The directory is removed only if it is empty.

### Restriction

When using OpenVMS format names, the *path* argument must be in the form *directory.dir*.

### Return Values

|    |                                                                     |
|----|---------------------------------------------------------------------|
| 0  | Indicates success.                                                  |
| −1 | An error occurred; <code>errno</code> is set to indicate the error. |

## sbrk

---

## sbrk

Determines the lowest virtual address that is not used with the program.

### Format

```
#include <unistd.h>

void *sbrk (long int incr);
```

### Argument

#### **incr**

The number of bytes to add to the current break address.

### Description

The `sbrk` function adds the number of bytes specified by its argument to the current break address and returns the old break address.

When a program is executed, the break address is set to the highest location defined by the program and data storage areas. Consequently, `sbrk` is needed only by programs that have growing data areas.

`sbrk(0)` returns the current break address.

### Return Values

|                           |                                                           |
|---------------------------|-----------------------------------------------------------|
| <code>x</code>            | The old break address.                                    |
| <code>(void *)(-1)</code> | Indicates that the program is requesting too much memory. |

### Restriction

Unlike other C library implementations, the *Compaq C* RTL memory allocation functions (such as `malloc`) do not rely on `brk` or `sbrk` to manage the program heap space. Consequently, on OpenVMS systems, calling `brk` or `sbrk` can interfere with memory allocation routines. The `brk` and `sbrk` functions are provided only for compatibility purposes.

---

**scalb** *(Alpha only)*

Returns the exponent of a floating-point number.

**Format**

```
#include <math.h>

double scalb (double x, double n);
float scalbf (float x, float n);
long double scalbl (long double x, long double n);
```

**Arguments**

**x**  
A nonzero floating-point number.

**n**  
An integer.

**Description**

The scalb functions return  $x \cdot (2^{**}n)$  for integer  $n$ .

**Return Values**

|                |                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------|
| <b>x</b>       | On successful completion, $x \cdot (2^{**}n)$ is returned.                                          |
| $\pm$ HUGE_VAL | On overflow, scalb returns $\pm$ HUGE_VAL (according to the sign of $x$ ) and sets errno to ERANGE. |
| 0              | Underflow occurred; errno is set to ERANGE.                                                         |
| $x$            | $x$ is $\pm$ Infinity.                                                                              |
| NaN            | $x$ or $n$ is NaN; errno is set to EDOM.                                                            |

## scanf

---

## scanf

Performs formatted input from the standard input (stdin), interpreting it according to the format specification. See Chapter 2 for information on format specifiers.

### Format

```
#include <stdio.h>

int scanf (const char *format_spec, ...);
```

### Arguments

#### **format\_spec**

Pointer to a string containing the format specification. The format specification consists of characters to be taken literally from the input or converted and placed in memory at the specified input sources. For a list of conversion characters, see Chapter 2.

...

Optional expressions that are pointers to objects whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, you can omit these input pointers. Otherwise, the function call must have at least as many input pointers as there are conversion specifications, and the conversion specifications must match the types of the input pointers.

Conversion specifications are matched to input sources in left-to-right order. Excess input pointers, if any, are ignored.

### Return Values

|     |                                                                                                                                                                                                                         |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x   | The number of successfully matched and assigned input items.                                                                                                                                                            |
| EOF | Indicates that a read error occurred prior to any successful conversions. The function sets <code>errno</code> . For a list of <code>errno</code> values set by this function, see <code>fscanf</code> in this section. |

---

## **[w]scanw**

Perform a `scanf` on the window. The `scanw` function acts on the `stdscr` window.

### **Format**

```
#include <curses.h>

int scanw (char *format_spec, ...);
int wscanw (WINDOW *win, char *format_spec, ...);
```

### **Arguments**

#### **win**

A pointer to the window.

#### **format\_spec**

A pointer to the format specification string.

...

Optional expressions that are pointers to objects whose resultant types correspond to conversion specifications given in the format specification. If no conversion specifications are given, you may omit these input pointers.

Otherwise, the function call must have at least as many input pointers as there are conversion specifications, and the conversion specifications must match the types of the input pointers.

Conversion specifications are matched to input sources in left-to-right order. Excess input pointers, if any, are ignored.

### **Description**

The formatting specification (*format\_spec*) and the other arguments are identical to those used with the `scanf` function.

The `scanw` and `wscanw` functions accept, format, and return a line of text from the terminal screen. For more information, see the `scrollok` and `scanf` functions in this section.

### **Return Values**

|     |                                                                                                  |
|-----|--------------------------------------------------------------------------------------------------|
| OK  | Indicates success.                                                                               |
| ERR | Indicates that the function makes the screen scroll illegally or that the scan was unsuccessful. |

## scroll

---

## scroll

Moves all the lines on the window up one line. The top line scrolls off the window and the bottom line becomes blank.

### Format

```
#include <curses.h>
int scroll (WINDOW *win);
```

### Argument

**win**  
A pointer to the window.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

---

## scrollok

Sets the scroll flag for the specified window.

### Format

```
#include <curses.h>

scrollok (WINDOW *win, bool boolf);
```

### Arguments

**win**

A pointer to the window.

**boolf**

A Boolean TRUE or FALSE value. If *boolf* is FALSE, scrolling is not allowed. This is the default setting. The *bool* type is defined in the `<curses.h>` header file as follows:

```
#define bool int
```

---

## seed48

Initializes a 48-bit random-number generator.

### Format

```
#include <stdlib.h>

unsigned short *seed48 (unsigned short seed_16v[3]);
```

### Arguments

#### **seed\_16v**

An array of three unsigned short int that form a 48-bit seed value.

### Description

The `seed48` function initializes the random-number generator. You can use this function in your program before calling the `drand48`, `lrand48`, or `mrand48` functions. (Although it is not recommended practice, constant default initializer values are supplied automatically if you call `drand48`, `lrand48`, or `mrand48` without calling an initialization function).

The `seed48` function works by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \bmod m \quad n > 0$$

The argument  $m$  equals  $2^{48}$ , so 48-bit integer arithmetic is performed. Unless you invoke the `lcong48` function, the multiplier value  $a$  and the addend value  $c$  are:

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

The initializer function `seed48`:

- Sets the value of  $X_i$  to the 48-bit value specified in the array pointed to by `seed_16v`.
- Returns a pointer to a 48-bit internal buffer that contains the previous value of  $X_i$ , used only by `seed48`.

The returned pointer allows you to restart the pseudorandom sequence at a given point. Use the pointer to copy the previous  $X_i$  value into a temporary array. To resume where the original sequence left off, you can call `seed48` with a pointer to this array.

See also `drand48`, `lrand48`, and `mrand48` in this section.

### Return Value

|   |                                        |
|---|----------------------------------------|
| x | A pointer to a 48-bit internal buffer. |
|---|----------------------------------------|



---

## seekdir

Sets the position of a directory stream.

### Format

```
#include <dirent.h>

void seekdir (DIR *dir_pointer, long int location);
```

### Arguments

**dir\_pointer**

A pointer to the `dir` structure of an open directory.

**location**

The number of an entry relative to the start of the directory.

### Description

The `seekdir` function sets the position of the next `readdir` operation on the directory stream specified by *dir\_pointer* to the position specified by *location*. The value of *location* should be returned from an earlier call to *telldir*.

If the value of *location* was not returned by a call to the `telldir` function, or if there was an intervening call to the `rewinddir` function on this directory stream, the effect is unspecified.

The type `DIR`, defined in the `<dirent.h>` header file, represents a directory stream. A directory stream is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files. You can remove files from or add files to a directory asynchronously to the operation of the `readdir` function.

See `readdir`, `rewinddir`, and `telldir` in this section.

## [w]setattr

---

## [w]setattr

Activate the video display attribute *attr* within the window. The `setattr` function acts on the `stdscr` window.

### Format

```
#include <curses.h>

int setattr (int attr);

int wsetattr (WINDOW *win, int attr);
```

### Arguments

#### **win**

A pointer to the window.

#### **attr**

One of a set of video display attributes, which are blinking, boldface, reverse video, and underlining, and are represented by the defined constants `_BLINK`, `_BOLD`, `_REVERSE`, and `_UNDERLINE`, respectively. You can set multiple attributes by separating them with a bitwise OR operator ( `|` ) as follows:

```
setattr(_BLINK | _UNDERLINE);
```

### Description

The `setattr` and `wsetattr` functions are specific to *Compaq C* for *OpenVMS* Systems and are not portable.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

---

## setbuf

Associates a new buffer with an input or output file and potentially modifies the buffering behavior.

### Format

```
#include <stdio.h>

void setbuf (FILE *file_ptr, char *buffer);
```

### Arguments

**file\_ptr**

A file pointer.

**buffer**

A pointer to a character array, or a NULL pointer.

### Description

You can use the setbuf function after the specified file is opened but before any I/O operations are performed.

If *buffer* is a NULL pointer, then the call is equivalent to a call to setvbuf with the same *file\_ptr*, a NULL *buffer* pointer, a buffering type of \_IONBF (no buffering), and a buffer size of 0.

If *buffer* is not a NULL pointer, then the call is equivalent to a call to setvbuf with the same *file\_ptr*, the same *buffer* pointer, a buffering type of \_IOFBF, and a buffer size given by the value BUFSIZ (defined in <stdio.h>). Therefore, use BUFSIZ to allocate the *buffer* argument used in the call to setbuf. For example:

```
#include <stdio.h>
.
.
.
char my_buf[BUFSIZ];
.
.
.
setbuf(stdout, my_buf);
.
.
.
```

User programs must not depend on the contents of *buffer* once I/O has been performed on the stream. The *Compaq C* RTL might or might not use *buffer* for any given I/O operation.

The setbuf function originally allowed programmers to substitute larger buffers in place of the system default buffers in obsolete versions of UNIX. The large default buffer sizes in modern implementations of C make the use of this function unnecessary most of the time. The setbuf function is retained in the ANSI C standard for compatibility with old programs. New programs should use setvbuf instead, because it allows the programmer to bind the buffer size at run time instead of compile time, and it returns a testable result value.

## setenv

---

## setenv

Inserts or resets the environment variable *name* in the current environment list.

### Format

```
#include <stdlib.h>

int setenv (const char *name, const char *value, int overwrite);
```

### Arguments

**name**

A variable name in the environment variable list.

**value**

The value for the environment variable.

**overwrite**

A value of 0 or 1 indicating whether to reset the environment variable, if it exists.

### Description

The `setenv` function inserts or resets the environment variable *name* in the current environment list. If the variable *name* does not exist in the list, it is inserted with the *value* argument. If the variable does exist, the *overwrite* argument is tested. When the *overwrite* argument value is:

- 0 (zero) – then the variable is not reset.
- 1 – then the variable is reset to *value*.

### Return Values

|    |                                                                                                                                   |
|----|-----------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                |
| –1 | Indicates an error. <code>errno</code> is set to <code>ENOMEM</code> —Not enough memory available to expand the environment list. |

---

## setgid

With POSIX IDs disabled, implemented for program portability and serves no function. It returns 0 (to indicate success).

With POSIX IDs enabled, sets the group IDs.

### Format

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

int setgid (_gid_t gid); (_DECC_V4_SOURCE)
gid_t setgid (gid_t gid); (not _DECC_V4_SOURCE)
```

### Argument

#### **gid**

The value to which you want the group IDs set.

### Description

The setgid function can be used with POSIX style identifiers enabled or disabled.

---

#### Note

Currently, POSIX style IDs are supported only by some OpenVMS versions done for specific government agencies, but will be integrated into future OpenVMS releases. OpenVMS Version 7.3-1 does not support POSIX style IDs, but it does support 32-bit identifiers.

---

With POSIX IDs disabled (the default), the setgid function is implemented for program portability and serves no function. It returns 0 (to indicate success).

With POSIX style IDs enabled:

- If the process has the IMPERSONATE privilege, the setgid function sets the real group ID, effective group ID, and the saved set-group-ID to *gid*.
- If the process does not have appropriate privileges but *gid* is equal to the real group ID or to the saved set-group-ID, then the setgid function sets the effective group ID to *gid*. The real group ID and saved set-group-ID remain unchanged.
- Any supplementary group IDs of the calling process remain unchanged.

### Return Values

0                                      Successful completion.

## setgid

–1

Indicates an error. The function sets *errno* to one of the following values:

- EINVAL – The value of the *gid* argument is invalid and not supported by the implementation.
- EPERM – The process does not have appropriate privileges and *gid* does not match the real group ID or the saved set-group-ID.

---

## setitimer

Sets the value of interval timers.

### Format

```
#include <time.h>

int setitimer (int which, struct itimerval *value, struct itimerval *ovalue);
```

### Arguments

#### which

The type of interval timer. The *Compaq C* RTL only supports ITIMER\_REAL.

#### value

A pointer to an `itimerval` structure whose members specify a timer interval and the time left to the end of the interval.

#### ovalue

A pointer to an `itimerval` structure whose members specify a current timer interval and the time left to the end of the interval.

### Description

The `setitimer` function sets the timer specified by *which* to the value specified by *value*, returning the previous value of the timer if *ovalue* is nonzero.

A timer value is defined by the `itimerval` structure:

```
struct itimerval {
 struct timeval it_interval;
 struct timeval it_value;
};
```

The value of the `itimerval` structure members are: as follows

| itimerval Member Value             | Meaning                                                                                |
|------------------------------------|----------------------------------------------------------------------------------------|
| <code>it_interval = 0</code>       | Disables a timer after its next expiration (Assumes <code>it_value</code> is nonzero). |
| <code>it_interval = nonzero</code> | Specifies a value used in reloading <code>it_value</code> when the timer expires.      |
| <code>it_value = 0</code>          | Disables a timer.                                                                      |
| <code>it_value = nonzero</code>    | Indicates the time to the next timer expiration.                                       |

Time values smaller than the resolution of the system clock are rounded up to this resolution.

The `getitimer` function provides one interval timer, defined in the `<time.h>` header file as `ITIMER_REAL`. This timer decrements in real time. When the timer expires, it delivers a `SIGALARM` signal.

## setitimer

---

### Note

---

The interaction between `setitimer` and any of `alarm`, `sleep`, or `usleep` is unspecified.

---

## Return Values

|    |                                                                     |
|----|---------------------------------------------------------------------|
| 0  | Indicates success.                                                  |
| −1 | An error occurred; <code>errno</code> is set to indicate the error. |



---

## setjmp

Provides a way to transfer control from a nested series of function invocations back to a predefined point without returning normally. It does not use a series of return statements. The `setjmp` function saves the context of the calling function in an environment buffer.

### Format

```
#include <setjmp.h>

int setjmp (jmp_buf env);
```

### Argument

#### env

The environment buffer, which must be an array of integers long enough to hold the register context of the calling function. The type `jmp_buf` is defined in the `<setjmp.h>` header file. The contents of the general-purpose registers, including the program counter (PC), are stored in the buffer.

### Description

When `setjmp` is first called, it returns the value 0. If `longjmp` is then called, naming the same environment as the call to `setjmp`, control is returned to the `setjmp` call as if it had returned normally a second time. The return value of `setjmp` in this second return is the value supplied by you in the `longjmp` call. To preserve the true value of `setjmp`, the function calling `setjmp` must not be called again until the associated `longjmp` is called.

The `setjmp` function preserves the hardware general purpose registers, and the `longjmp` function restores them. After a `longjmp`, all variables have their values as of the time of the `longjmp` except for local automatic variables not marked volatile. These variables have indeterminate values.

The `setjmp` and `longjmp` functions rely on the OpenVMS condition-handling facility to effect a nonlocal goto with a signal handler. The `longjmp` function is implemented by generating a *Compaq C* RTL specified signal that allows the OpenVMS condition-handling facility to unwind back to the desired destination.

The *Compaq C* RTL must be in control of signal handling for any *Compaq C* image. For *Compaq C* to be in control of signal handling, you must establish all exception handlers through a call to the `VAXC$ESTABLISH` function. See Section 4.2.5 and the `VAXC$ESTABLISH` function in this section for more information.

---

#### Note

---

There are Alpha specific, nonstandard `decc$setjmp` and `decc$fast_longjmp` functions. To use these nonstandard functions instead of the standard ones, a program must be compiled with `__FAST_SETJMP` or `__UNIX_SETJMP` macros defined.

Unlike the standard `longjmp` function, the `decc$fast_longjmp` function does not convert its second argument from 0 to 1. After a call to `decc$fast_longjmp`, a corresponding `setjmp` function returns with the

## setjmp

exact value of the second argument specified in the `decc$fast_longjmp` call.

---

### Restrictions

You cannot invoke the `longjmp` function from an OpenVMS condition handler. However, you may invoke `longjmp` from a signal handler that has been established for any signal supported by the *Compaq C* RTL, subject to the following nesting restrictions:

- The `longjmp` function will not work if you invoke it from nested signal handlers. The result of the `longjmp` function, when invoked from a signal handler that has been entered as a result of an exception generated in another signal handler, is undefined.
- Do not invoke the `setjmp` function from a signal handler unless the associated `longjmp` is to be issued before the handling of that signal is completed.
- Do not invoke the `longjmp` function from within an exit handler (established with `atexit` or `SYS$DCLEXH`). Exit handlers are invoked after image tear-down, so the destination address of the `longjmp` no longer exists.
- Invoking `longjmp` from within a signal handler to return to the main thread of execution might leave your program in an inconsistent state. Possible side effects include the inability to perform I/O or to receive any more UNIX signals. Use `siglongjmp` instead.

### Return Values

See the Description section.

---

## setlocale

Selects the appropriate portion of the program's locale as specified by the *category* and *locale* arguments. You can use this function to change or query one category or the program's entire current locale.

### Format

```
#include <locale.h>

char *setlocale (int category, const char *locale);
```

### Arguments

#### **category**

The name of the category. Specify LC\_ALL to change or query the entire locale. Other valid category names are:

- LC\_COLLATE
- LC\_CTYPE
- LC\_MESSAGES
- LC\_MONETARY
- LC\_NUMERIC
- LC\_TIME

#### **locale**

Pointer to a string that specifies the locale.

### Description

The setlocale function sets or queries the appropriate portion of the program's locale as specified by the *category* and *locale* arguments. Specifying LC\_ALL for the category argument names the entire locale; specifying the other values name only a portion of the program's locale.

The *locale* argument points to a character string that identifies the locale to be used. This argument can be one of the following:

- Name of the public locale

Specifies the public locale in the following format:

```
language_country.codeset[@modifier]
```

The function searches for the public locale binary file in the location defined by the logical name SYS\$I18N\_LOCALE. The file type defaults to .LOCALE. The period (.) and at-sign (@) characters in the name are replaced by an underscore (\_).

For example:

If the specified name is "zh\_CN.dechanzi@radical", the function searches for the SYS\$I18N\_LOCALE:ZH\_CN\_DECHANZI\_RADICAL.LOCALE binary locale file.

- A file specification

## setlocale

Specifies the binary locale file. It can be any valid file specification. If either the device or directory is omitted, the function first applies the current caller's device and directory as defaults for any missing component. If the file not found, the function applies the device and directory defined by the SYS\$I18N\_LOCALE logical name as defaults. The file type defaults to .LOCALE.

No wildcards are allowed. The binary locale file cannot reside on a remote node.

- "C"

Specifies the C locale. If a program does not call setlocale, the C locale is the default.

- "POSIX"

This is the same as the C locale.

- ""

Specifies that the locale is initialized from the setting of the international environment logical names. The function checks the following logical names in the order shown until it finds a logical that is defined:

1. LC\_ALL
2. Logical names corresponding to the category. For example, if LC\_NUMERIC is specified as the category, then the first logical name that setlocale checks is LC\_NUMERIC.
3. LANG
4. SYS\$LC\_ALL
5. The system default for the category, which is defined by the SYS\$LC\_\* logical names. For example, the default for the LC\_NUMERIC category is defined by the SYS\$LC\_NUMERIC logical name.
6. SYS\$LANG

If none of the logical names is defined, the C locale is used as the default. The SYS\$LC\_\* logical names are set up at the system startup time.

Like the *locale* argument, the equivalence name of the international environment logical name can be either the name of the public locale or the file specification. The setlocale function treats this equivalence name as if it were specified as the *locale* argument.

- NULL

Causes setlocale to query the current locale. The function returns a pointer to a string describing the portion of the program's locale associated with *category*. Specifying the LC\_ALL category returns the string describing the entire locale. The locale is not changed.

- The string returned from the previous call to setlocale

Causes the function to restore the portion of the program's locale associated with *category*. If the string contains the description of the entire locale, the part of the string corresponding to *category* is used. If the string describes the portion of the program's locale for a single category, this locale is used. For example, this means that you can use the string returned from the call setlocale with the LC\_COLLATE category to set the same locale for the LC\_MESSAGES category.

If the specified locale is available, then `setlocale` returns a pointer to the string that describes the portion of the program's locale associated with *category*. For the `LC_ALL` category, the returned string describes the entire program's locale. If an error occurs, a `NULL` pointer is returned and the program's locale is not changed.

Subsequent calls to `setlocale` overwrite the returned string. If that part of the locale needs to be restored, the program should save the string. The calling program should make no assumptions about the format or length of the returned string.

## Return Values

|                   |                                                         |
|-------------------|---------------------------------------------------------|
| <code>x</code>    | Pointer to a string describing the locale.              |
| <code>NULL</code> | Indicates an error occurred; <code>errno</code> is set. |

## Example

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

/* This program calls setlocale() three times. The second call */
/* is for a nonexistent locale. The third call is for an */
/* existing file that is not a locale file. */
main()
{
 char *ret_str;
 errno = 0;
 printf("setlocale (LC_ALL, \"POSIX\")");
 ret_str = (char *) setlocale(LC_ALL, "POSIX");
 if (ret_str == NULL)
 perror("setlocale error");
 else
 printf(" call was succesfull\n");
 errno = 0;
 printf("\n\nsetlocale (LC_ALL, \"junk.junk_codeset\")");
 ret_str = (char *) setlocale(LC_ALL, "junk.junk_codeset");
 if (ret_str == NULL)
 perror(" returned error");
 else
 printf(" call was succesfull\n");
 errno = 0;
 printf("\n\nsetlocale (LC_ALL, \"sys$login:login.com\")");
 ret_str = (char *) setlocale(LC_ALL, "sys$login:login.com");
 if (ret_str == NULL)
 perror(" returned error");
 else
 printf(" call was succesfull\n");
}
```

Running the example program produces the following result:

## setlocale

```
setlocale (LC_ALL, "POSIX") call was succesfull
setlocale (LC_ALL, "junk.junk_codeset")
returned error: no such file or directory
setlocale (LC_ALL, "sys$login:login.com")
returned error: nontranslatable vms error code: 0x35C07C
%c-f-localebad, not a locale file
```

---

## setpwent

Accesses the first entry of user attribute information in the user database.

### Format

```
#include <stdio.h>
#include <pwd.h>
int *setpwent (void);
```

### Description

Use the `setpwent` function to access basic user attributes about a specified user. The `setpwent` function ensures that the next call to the `getpwent` function returns the first entry.

See also `getpwent` in this section.

## setstate

---

## setstate

Restarts, and changes random-number generators.

### Format

```
char *setstate (char *state;)
```

### Argument

#### **state**

Points to the array of state information.

### Description

The `setstate` function handles restarting and changing random-number generators.

Once you initialize a state, the `setstate` function allows rapid switching between state arrays. The array defined by *state* is used for further random-number generation until the `initstate` function is called or the `setstate` function is called again. The `setstate` function returns a pointer to the previous state array.

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

- Use the `initstate` function, with the desired *seed*, state array, and size of the array.
- Use the `setstate` function, with the desired state, followed by the `srandom` function with the desired seed. The advantage of using both functions is that you do not have to save the state array size once you initialize it.

See also `initstate`, `srandom`, and `random` in this section.

### Return Values

- |   |                                                                                                                                                                                                                                |
|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x | A pointer to the previous state array information.                                                                                                                                                                             |
| 0 | Indicates an error. The state information is damaged. Further specified in the following <code>errno</code> value: <ul style="list-style-type: none"><li>• <code>EINVAL</code>—The <i>state</i> argument is invalid.</li></ul> |



---

## setuid

With POSIX IDs disabled, implemented for program portability and serves no function. It returns 0 (to indicate success).

With POSIX IDs enabled, sets the user IDs.

### Format

```
#include <types.h>

#include <unistd.h>

int setuid (__uid_t uid); (_DECC_V4_SOURCE)

uid_t setuid (uid_t uid); (not _DECC_V4_SOURCE)
```

### Argument

#### **uid**

The value to which you want the user IDs set.

### Description

The setuid function can be used with POSIX style identifiers enabled or disabled.

---

#### **Note**

Currently, POSIX style IDs are supported only by some OpenVMS versions done for specific government agencies, but will be integrated into future OpenVMS releases. OpenVMS Version 7.3-1 does not support POSIX style IDs, but it does support 32-bit identifiers.

---

With POSIX IDs disabled (the default), the setuid function is implemented for program portability and serves no function. It returns 0 (to indicate success).

With POSIX style IDs enabled:

- If the process has the IMPERSONATE privilege, the setuid function sets the real user ID, effective user ID, and the saved set-user-ID to *uid*.
- If the process does not have appropriate privileges but *uid* is equal to the real user ID or to the saved set-user-ID, then the setuid function sets the effective user ID to *uid*. The real user ID and saved set-user-ID remain unchanged.

### Return Values

|   |                        |
|---|------------------------|
| 0 | Successful completion. |
|---|------------------------|

## setuid

–1

Indicates an error. The function sets *errno* to one of the following values:

- EINVAL – The value of the *uid* argument is invalid and not supported by the implementation.
- EPERM – The process does not have appropriate privileges and *uid* does not match the real user ID or the saved set-user-ID.

---

## setvbuf

Associates a buffer with an input or output file and potentially modifies the buffering behavior.

### Format

```
#include <stdio.h>

int setvbuf (FILE *file_ptr, char *buffer, int type, size_t size);
```

### Arguments

**file\_ptr**

A pointer to a file.

**buffer**

A pointer to a character array, or a NULL pointer.

**type**

The buffering type. Use one of the following values defined in <stdio.h>: `_IONBF`, `_IOFBF`, `_IOLBF`.

**size**

The number of bytes to be used in *buffer* by the *Compaq C* RTL for buffering this file. The buffer size must be a minimum of 8192 bytes and a maximum of 32767 bytes.

### Description

You can use the `setvbuf` function after the file is opened but before any I/O operations are performed.

The ANSI C standard defines the following types of file buffering. In unbuffered I/O, each I/O operation is performed immediately. Output characters or lines are written to the output device before control is returned to the program. Input characters or lines are sent directly to the program without read-ahead by the *Compaq C* RTL.

In line-buffered I/O, characters are buffered in an area of memory until a new-line character is seen, at which point the appropriate RMS routine is called to transmit the entire buffer. Line buffering is more efficient than unbuffered I/O since it reduces the system overhead, but it delays the availability of the data to the user or disk on output.

In fully buffered I/O, characters are buffered in an area of memory until the buffer is full, regardless of the presence of break characters. Full buffering is more efficient than line buffering or unbuffered I/O, but it delays the availability of output data even longer than line buffering.

Use the values `_IONBF`, `_IOLBF`, and `_IOFBF` defined in <stdio.h> for the *type* argument to specify unbuffered, line-buffered, and fully buffered I/O, respectively.

If `_IONBF` is specified for *type*, I/O will be unbuffered and the *buffer* and *size* arguments are ignored.

## setvbuf

If `_IOLBF` or `_IOFBF` is specified for `type`, the *Compaq C* RTL will use line-buffered I/O if `file_ptr` specifies a terminal device; otherwise, it will use fully buffered I/O.

The *Compaq C* RTL automatically allocates a buffer to use for each I/O stream. So there are several buffer allocation possibilities:

- If `buffer` is not a NULL pointer and `size` is not smaller than the automatically allocated buffer, then `setvbuf` uses `buffer` as the file buffer.
- If `buffer` is a NULL pointer or `size` is smaller than the automatically allocated buffer, the automatically allocated buffer is used as the buffer area.
- If `buffer` is a NULL pointer and `size` is larger than the automatically allocated buffer, then `setvbuf` allocates a new buffer equal to the specified size and uses that as the file buffer.

User programs must not depend on the contents of `buffer` once I/O has been performed on the stream. The *Compaq C* RTL might or might not use `buffer` for any given I/O operation.

Generally, it is unnecessary to use `setvbuf` or `setbuf` to control the buffer size used by the *Compaq C* RTL. The automatically allocated buffer sizes are chosen for efficiency based on the kind of I/O operations performed and the device characteristics (such as terminal, disk, or socket).

The `setvbuf` and `setbuf` functions are useful to introduce buffering for improved performance when writing a large amount of text to the `stdout` stream. This stream is unbuffered by default when bound to a terminal device (the normal case), and therefore incurs a large number of OpenVMS buffered I/O operations unless *Compaq C* RTL buffering is introduced by a call to `setvbuf` or `setbuf`.

The `setvbuf` function is used only to control the buffering used by the *Compaq C* RTL, not the buffering used by the underlying RMS I/O operations. You can modify RMS default buffering behavior by specifying various values for the `ctx`, `fop`, `rat`, `gbc`, `mbc`, `mbf`, `rfm`, and `rop` RMS keywords when the file is opened by the `creat`, `freopen` or `open` functions.

## Return Values

|               |                                                                                                                                                                                            |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0             | Indicates success.                                                                                                                                                                         |
| nonzero value | Indicates that an invalid input value was specified for <code>type</code> or <code>file_ptr</code> , or because <code>file_ptr</code> is being used by another thread (see Section 1.9.1). |

---

## sigaction

Specifies the action to take upon delivery of a signal.

### Format

```
#include <signal.h>

int sigaction (int sig, const struct sigaction *action, struct sigaction *o_action);
```

### Arguments

#### **sig**

The signal for which the action is to be taken.

#### **action**

A pointer to a `sigaction` structure that describes the action to take when you receive the signal specified by the *sig* argument.

#### **o\_action**

A pointer to a `sigaction` structure. When the `sigaction` function returns from a call, the action previously attached to the specified signal is stored in this structure.

### Description

When a process requests the `sigaction` function, the process can both examine and specify what action to perform when the specified signal is delivered. The arguments determine the behavior of the `sigaction` function as follows:

- Specifying the *sig* argument identifies the affected signal. Use any one of the signal values defined in the `<signal.h>` header file, except `SIGKILL`.  
If *sig* is `SIGCHLD` and the `SA_NOCLDSTOP` flag is not set in `sa_flags`, then a `SIGCHLD` signal is generated for the calling process whenever any of its child processes stop. If *sig* is `SIGCHLD` and the `SA_NOCLDSTOP` flag is set in `sa_flags`, then `SIGCHLD` signal is not generated in this way.
- Specifying the *action* argument, if not null, points to a `sigaction` structure that defines what action to perform when the signal is received. If the *action* argument is null, signal handling remains unchanged, so you can use the call to inquire about the current handling of the signal.
- Specifying the *o\_action* argument, if not null, points to a `sigaction` structure that contains the action previously attached to the specified signal.

The `sigaction` structure consists of the following members:

```
void (*sa_handler)(int);
sigset_t sa_mask;
int sa_flags;
```

The `sigaction` structure members are defined as follows:

## sigaction

|                         |                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>sa_handler</code> | This member can contain the following values: <ul style="list-style-type: none"><li>• <code>SIG_DFL</code> – Specifies the default action taken when the signal is delivered.</li><li>• <code>SIG_IGN</code> – Specifies that the signal has no effect on the receiving process.</li><li>• Function pointer – Requests to catch the signal. The signal causes the function call.</li></ul> |
| <code>sa_mask</code>    | This member can request that individual signals, in addition to those in the process signal mask, are blocked from delivery while the signal handler function specified by the <i>sa_handler</i> member is executing.                                                                                                                                                                      |
| <code>sa_flags</code>   | This member can set the flags to enable further control over the actions taken when a signal is delivered.                                                                                                                                                                                                                                                                                 |

The `sa_flags` member of the `sigaction` structure has the following values:

|                           |                                                                                                                                                                                                                                                                             |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>SA_ONSTACK</code>   | Setting this bit causes the system to run the signal catching function on the signal stack specified by the <code>sigstack</code> function. If this bit is not set, the function runs on the stack of the process where the signal is delivered.                            |
| <code>SA_RESETHAND</code> | Setting this bit resets the signal to <code>SIG_DFL</code> . Be aware that you cannot automatically reset <code>SIGILL</code> and <code>SIGTRAP</code> .                                                                                                                    |
| <code>SA_NODEFER</code>   | Setting this bit does not automatically block the signal as it is caught.                                                                                                                                                                                                   |
| <code>SA_NOCLDSTOP</code> | If this bit is set and the <i>sig</i> argument is equal to <code>SIGCHLD</code> and a child process of the calling process stops, then a <code>SIGCHLD</code> signal is sent to the calling process only if <code>SA_NOCLDSTOP</code> is not set for <code>SIGCHLD</code> . |

When a signal is caught by a signal-catching function installed by `sigaction`, a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either `sigprocmask` or `sigsuspend` is made. This mask is formed by taking the union of the current signal mask and the value of the `sa_mask` for the signal being delivered unless `SA_NODEFER` or `SA_RESETHAND` is set, and then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.

Once an action is installed for a specific signal, it remains installed until another action is explicitly requested (by another call to `sigaction`), until the `SA_RESETHAND` flag causes resetting of the handler, or until one of the `exec` functions is called.

If the previous action for a specified signal had been established by `signal`, the values of the fields returned in the structure pointed to by the *o\_action* argument of `sigaction` are unspecified, and in particular *o\_action*->`sa_handler` is not necessarily the same value passed to `signal`. However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to `sigaction` by means of the *action* argument of `sigaction`, the signal is handled as if the original call to `signal` were repeated.

If `sigaction` fails, no new signal handler is installed.

It is unspecified whether an attempt to set the action for a signal that cannot be caught or ignored to SIG\_DFL is ignored or causes an error to be returned with `errno` set to EINVAL.

See Section 4.2 for more information on signal handling.

---

**Note**

---

The `sigvec` and `signal` functions are provided for compatibility to old UNIX systems; their function is a subset of that available with the `sigaction` function.

---

See also `sigstack`, `sigvec`, `signal`, `wait`, `read`, and `write` in this section.

## Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| -1 | Indicates an error; A new signal handler is not installed. <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"><li>• EFAULT – The <i>action</i> or <i>o_action</i> argument points to a location outside of the allocated address space of the process.</li><li>• EINVAL – The <i>sig</i> argument is not a valid signal number. Or an attempt was made to ignore or supply a handler for the SIGKILL, SIGSTOP, and SIGCONT signals.</li></ul> |

## sigaddset

---

## sigaddset

Adds the specified individual signal.

### Format

```
#include <signal.h>

int sigaddset (sigset_t *set, int sig_number);
```

### Arguments

**set**  
The signal set.

**sig\_number**  
The individual signal.

### Description

The `sigaddset` function manipulates sets of signals. This function operates on data objects that you can address by the application, not on any set of signals known to the system. For example, this function does not operate on the set blocked from delivery to a process or the set pending for a process.

The `sigaddset` function adds the individual signal specified by *sig\_number* from the signal set specified by *set*.

### Example

The following example shows how to generate and use a signal mask that blocks only the SIGINT signal from delivery.

```
#include <signal.h>
int return_value;
sigset_t newset;
. . .
sigemptyset(&newset);
sigaddset(&newset, SIGINT);
return_value = sigprocmask (SIG_SETMASK, &newset, NULL);
```

### Return Values

- |    |                                                                                                                                      |
|----|--------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                   |
| -1 | Indicates an error; <code>errno</code> is set to one of the following values:                                                        |
|    | <ul style="list-style-type: none"><li>• <code>EINVAL</code> – The value of <i>sig_number</i> is not a valid signal number.</li></ul> |



## Format

## Argument

### Description

You can express signals in mnemonics (such as SIGBUS for a protection violation) or numbers as defined in the `<signal.h>` header file, and you can express combinations of signals by using the bitwise OR operator (`|`).

## Return Value

REF-437

## sigdelset

---

## sigdelset

Deletes a specified individual signal.

### Format

```
#include <signal.h>

int sigdelset (sigset_t *set, int sig_number,)
```

### Arguments

**set**  
The signal set.

**sig\_number**  
The individual signal.

### Description

The `sigdelset` function deletes the individual signal specified by *sig\_number* from the signal set specified by *set*.

This function operates on data objects that you can address by the application, not on any set of signals known to the system. For example, this function does not operate on the set blocked from delivery to a process or the set pending for a process.

### Return Values

- |    |                                                                                                                                      |
|----|--------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                   |
| -1 | Indicates an error; <code>errno</code> is set to one of the following values:                                                        |
|    | <ul style="list-style-type: none"><li>• <code>EINVAL</code> – The value of <i>sig_number</i> is not a valid signal number.</li></ul> |

---

## sigemptyset

Initializes the signal set to exclude all signals.

### Format

```
#include <signal.h>

int sigemptyset (sigset_t *set);
```

### Argument

**set**  
The signal set.

### Description

The `sigemptyset` function initializes the signal set pointed to by *set* such that you exclude all signals. A call to `sigemptyset` or `sigfillset` must be made at least once for each object of type `sigset_t` prior to any other use of that object.

This function operates on data objects that you can address by the application, not on any set of signals known to the system. For example, this function does not operate on the set blocked from delivery to a process or the set pending for a process.

See also `sigfillset` in this section.

### Example

The following example shows how to generate and use a signal mask that blocks only the SIGINT signal from delivery.

```
#include <signal.h>
int return_value;
sigset_t newset;
. . .
sigemptyset(&newset);
sigaddset(&newset, SIGINT);
return_value = sigprocmask (SIG_SETMASK, &newset, NULL);
```

### Return Values

|    |                                                                                 |
|----|---------------------------------------------------------------------------------|
| 0  | Indicates success.                                                              |
| −1 | Indicates an error; the global <code>errno</code> is set to indicate the error. |

## sigfillset

---

## sigfillset

Initializes the signal set to include all signals.

### Format

```
#include <signal.h>

int sigfillset (sigset_t *set);
```

### Argument

**set**  
The signal set.

### Description

The `sigfillset` function initializes the signal set pointed to by *set* such that you include all signals. A call to `sigemptyset` or `sigfillset` must be made at least once for each object of type `sigset_t` prior to any other use of that object.

This function operates on data objects that you can address by the application, not on any set of signals known to the system. For example, this function does not operate on the set blocked from delivery to a process or the set pending for a process.

See also `sigemptyset` in this section.

### Return Values

- |    |                                                                                                                                                   |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                                |
| −1 | Indicates an error; <code>errno</code> is set to one of the following values:                                                                     |
|    | <ul style="list-style-type: none"><li>• <code>EINVAL</code> – The value of the <i>sig_number</i> argument is not a valid signal number.</li></ul> |

---

## sigismember

Tests whether a specified signal is a member of the signal set.

### Format

```
#include <signal.h>

int sigismember (const sigset_t *set, int sig_number);
```

### Arguments

**set**

The signal set.

**sig\_number**

The individual signal.

### Description

The `sigismember` function tests whether *sig\_number* is a member of the signal set pointed to by *set*.

This function operates on data objects that you can address by the application, not on any set of signals known to the system. For example, this function does not operate on the set blocked from delivery to a process or the set pending for a process.

### Return Values

|   |                                                                                |
|---|--------------------------------------------------------------------------------|
| 1 | Indicates success. The specified signal is a member of the specified set.      |
| 0 | Indicates an error. The specified signal is not a member of the specified set. |

## siglongjmp

---

## siglongjmp

Nonlocal goto with signal handling.

### Format

```
#include <setjmp.h>

void siglongjmp (sigjmp_buf env, int value);
```

### Arguments

**env**

An address for a `sigjmp_buf` structure.

**value**

A nonzero value.

### Description

The `siglongjmp` function restores the environment saved by the most recent call to `sigsetjmp` in the same process with the corresponding `sigjmp_buf` argument.

All accessible objects have values when `siglongjmp` is called, with one exception: values of objects of automatic storage duration that changed between the `sigsetjmp` call and `siglongjmp` call are indeterminate.

Because it bypasses the usual function call and return mechanisms, `siglongjmp` executes correctly during interrupts, signals, and any of their associated functions. However, if you invoke `siglongjmp` from a nested signal handler (for example, from a function invoked as a result of a signal raised during the handling of another signal), the behavior is undefined.

The `siglongjmp` function restores the saved signal mask only if you initialize the `env` argument by a call to `sigsetjmp` with a nonzero *savemask* argument.

After `siglongjmp` is completed, program execution continues as if the corresponding call of `sigsetjmp` just returned the value specified by *value*.

The `siglongjmp` function cannot cause `sigsetjmp` to return 0 (zero); if *value* is 0, `sigsetjmp` returns 1

See also `sigsetjmp` in this section.

---

## sigmask

Constructs the mask for a given signal number.

### Format

```
#include <signal.h>
int sigmask (signal);
```

### Argument

**signal**

The signal number for which the mask is to be constructed.

### Description

The sigmask function is used to construct the mask for a given *signal*. This mask can be used with the sigblock function.

### Return Value

|   |                                        |
|---|----------------------------------------|
| x | The mask constructed for <i>signal</i> |
|---|----------------------------------------|

## signal

---

## signal

Allows you to specify the way in which the signal *sig* is to be handled: use the default handling for the signal, ignore the signal, or call the signal handler at the address specified.

### Format

```
#include <signal.h>

void (*signal (int sig, void (*func) (int))) (int);
```

### Arguments

#### **sig**

The number or mnemonic associated with a signal. This argument is usually one of the mnemonics defined in the `<signal.h>` header file.

#### **func**

Either the action to take when the signal is raised, or the address of a function needed to handle the signal.

### Description

If *func* is the constant `SIG_DFL`, the action for the given signal is reset to the default action, which is to terminate the receiving process. If the argument is `SIG_IGN`, the signal is ignored. Not all signals can be ignored.

If *func* is neither `SIG_DFL` nor `SIG_IGN`, it specifies the address of a signal-handling function. When the signal is raised, the addressed function is called with *sig* as its argument. When the addressed function returns, the interrupted process continues at the point of interruption. (This is called catching a signal. Signals are reset to `SIG_DFL` after they are caught, except as shown in Chapter 4.)

You must call the signal function each time you want to catch a signal.

See Section 4.2 for more information on signal handling.

To cause a OpenVMS exception or a signal to generate a UNIX style signal, user OpenVMS condition handlers must return `SS$_RESIGNAL` upon receiving any exception that they do not want to handle. Returning `SS$_CONTINUE` prevents the correct generation of a UNIX style signal. See Chapter 4 for a list of OpenVMS exceptions that correspond to UNIX signals.

### Return Values

|         |                                                                          |
|---------|--------------------------------------------------------------------------|
| x       | The address of the function previously established to handle the signal. |
| SIG_ERR | Indicates that the <i>sig</i> argument is out of range.                  |



---

## sigpause

Assigns *mask* to the current set of masked signals and then waits for a signal.

### Format

```
#include <signal.h>
int sigpause (int mask);
```

### Argument

**mask**  
The signals to be blocked.

### Description

See the sigblock function in this section for information about the *mask* argument.

When control returns to sigpause, the function restores the previous set of masked signals, sets `errno` to `EINTR`, and returns `-1` to indicate an interrupt. The value `EINTR` is defined in the `<errno.h>` header file.

### Return Value

|                 |                                                                           |
|-----------------|---------------------------------------------------------------------------|
| <code>-1</code> | Indicates an interrupt. <code>errno</code> is set to <code>EINTR</code> . |
|-----------------|---------------------------------------------------------------------------|

## sigpending

---

## sigpending

Examines pending signals.

### Format

```
#include <signal.h>

int sigpending (sigset_t *set);
```

### Argument

**set**  
A pointer to a `sigset_t` structure.

### Description

The `sigpending` function stores the set of signals that are blocked from delivery and pending to the calling process in the location pointed to by the *set* argument.

Call either the `sigemptyset` or the `sigfillset` function at least once for each object of type `sigset_t` prior to any other use of that object. If you do not initialize an object in this way and supply an argument to the `sigpending` function, the result is undefined.

See also `sigemptyset` and `sigfillset` in this section.

### Return Values

- |    |                                                                                |
|----|--------------------------------------------------------------------------------|
| 0  | Indicates success.                                                             |
| −1 | Indicates an error; <code>errno</code> is set to the following value:          |
|    | <ul style="list-style-type: none"><li>• SIGSEGV – Bad mask argument.</li></ul> |

---

## sigprocmask

Sets the current signal mask.

### Format

```
#include <signal.h>

int sigprocmask (int how, const sigset_t *set, sigset_t *o_set);
```

### Arguments

#### **how**

An integer value that indicates how to change the set of masked signals. Use one of the following values:

|             |                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------|
| SIG_BLOCK   | The resulting set is the union of the current set and the signal set pointed to by the <i>set</i> argument.                          |
| SIG_UNBLOCK | The resulting set is the intersection of the current set and the complement of the signal set pointed to by the <i>set</i> argument. |
| SIG_SETMASK | The resulting set is the signal set pointed to by the <i>set</i> argument.                                                           |

#### **set**

The signal set. If the value of the *set* argument is:

- Not NULL – It points to a set of signals used to change the currently blocked set.
- NULL – The value of the *how* argument is not significant, and the process signal mask is unchanged, so you can use the call to inquire about currently blocked signals.

#### **o\_set**

A non-NULL pointer to the location where the signal mask in effect at the time of the call is stored.

### Description

The sigprocmask function is used to examine or change the signal mask of the calling process.

Typically, use the sigprocmask SIG\_BLOCK value to block signals during a critical section of code, then use the sigprocmask SIG\_SETMASK value to restore the mask to the previous value returned by the sigprocmask SIG\_BLOCK value.

If there are any unblocked signals pending after the call to the sigprocmask function, at least one of those signals is delivered before the sigprocmask function returns.

You cannot block SIGKILL or SIGSTOP signals with the sigprocmask function. If a program attempts to block one of these signals, the sigprocmask function gives no indication of the error.

## sigprocmask

### Example

The following example shows how to set the signal mask to block only the SIGINT signal from delivery:

```
#include <signal.h>

int return_value;
sigset_t newset;
. . .
sigemptyset(&newset);
sigaddset(&newset, SIGINT);
return_value = sigprocmask (SIG_SETMASK, &newset, NULL);
```

### Return Values

0

Indicates success.

−1

Indicates an error. The signal mask of the process is unchanged. `errno` is set to one of the following values:

- **EINVAL** – The value of the *how* argument is not equal to one of the defined values.
- **EFAULT** – The *set* or *o\_set* argument points to a location outside the allocated address space of the process.

---

## sigsetjmp

Sets jump point for a nonlocal goto.

### Format

```
#include <setjmp.h>

init sigsetjmp (sigjmp_buf env, int savemask);
```

### Arguments

**env**

An address for a sigjmp\_buf structure.

**savemask**

An integer value that specifies whether you need to save the current signal mask.

### Description

The sigsetjmp function saves its calling environment in its *env* argument for later use by the siglongjmp function.

If the value of *savemask* is not 0 (zero), sigsetjmp also saves the process' current signal mask as part of the calling environment.

See also siglongjmp in this section.

### Restrictions

You cannot invoke the longjmp function from an OpenVMS condition handler. However, you may invoke longjmp from a signal handler that has been established for any signal supported by the *Compaq C* RTL, subject to the following nesting restrictions:

- The longjmp function will not work if you invoke it from nested signal handlers. The result of the longjmp function, when invoked from a signal handler that has been entered as a result of an exception generated in another signal handler, is undefined.
- Do not invoke the sigsetjmp function from a signal handler unless the associated longjmp is to be issued before the handling of that signal is completed.
- Do not invoke the longjmp function from within an exit handler (established with atexit or SYS\$DCLEXH). Exit handlers are invoked after image tear-down, so the destination address of the longjmp no longer exists.
- Invoking longjmp from within a signal handler to return to the main thread of execution might leave your program in an inconsistent state. Possible side effects include the inability to perform I/O or to receive any more UNIX signals. Use siglongjmp instead.

## **sigsetjmp**

### **Return Values**

0

Indicates success.

nonzero

The return is a call to the siglongjmp function.



---

**sigstack** *(VAX only)*

Defines an alternate stack on which to process signals. This allows the processing of signals in a separate environment from that of the current process. This function is nonreentrant.

**Format**

```
#include <signal.h>

int sigstack (struct sigstack *ss, struct sigstack *oss);
```

**Arguments****ss**

If *ss* is not NULL, it specifies the address of a structure that holds a pointer to a designated section of memory to be used as a signal stack on which to deliver signals.

**oss**

If *oss* is not NULL, it specifies the address of a structure in which the old value of the stack address is returned.

**Description**

The `sigstack` structure is defined in the standard header file `<signal.h>` as follows:

```
struct sigstack
{
 char *ss_sp;
 int ss_onstack;
};
```

If the `sigvec` function specifies that the signal handler is to execute on the signal stack, the system checks to see if the process is currently executing on that stack. If the process is not executing on the signal stack, the system arranges a switch to the signal stack for the duration of the signal handler's execution. If the *oss* argument is not NULL, the current state of the signal stack is returned.

Signal stacks must be allocated an adequate amount of storage; they do not expand like the run-time stack. For example, if your signal handler calls `printf` or any similarly complex *Compaq C* RTL routine, at least 12,000 bytes of storage should be allocated for the signal stack. If the stack overflows, an error occurs.

*ss\_sp must* point to at least four bytes before the end of the allocated memory area (see the example). This is architecture-dependent and possibly not portable to other machine architectures or operating systems.

The `sigstack` structure is defined in the `<signal.h>` header file.

**Return Values**

|    |                    |
|----|--------------------|
| 0  | Indicates success. |
| -1 | Indicates failure. |



**Example**

```
#define ss_size 15000
static char mystack[ss_size];
struct sigstack ss = {&mystack + sizeof(mystack) - sizeof(void *), 1};
```

---

## sigsuspend

Atomically changes the set of blocked signals and waits for a signal.

### Format

```
#include <signal.h>

int sigsuspend (const sigset_t *signal_mask);
```

### Argument

#### **signal\_mask**

A pointer to a set of signals.

### Description

The `sigsuspend` function replaces the signal mask of the process with the set of signals pointed to by the *signal\_mask* argument. Then it suspends execution of the process until delivery of a signal whose action is either to execute a signal catching function or to terminate the process. You cannot block the SIGKILL or SIGSTOP signals with the `sigsuspend` function. If a program attempts to block either of these signals, `sigsuspend` gives no indication of the error.

If delivery of a signal causes the process to terminate, `sigsuspend` does not return. If delivery of a signal causes a signal catching function to execute, `sigsuspend` returns after the signal catching function returns, with the signal mask restored to the set that existed prior to the call to `sigsuspend`.

The `sigsuspend` function sets the signal mask and waits for an unblocked signal as one atomic operation. This means that signals cannot occur between the operations of setting the mask and waiting for a signal. If a program invokes `sigprocmask` SIG\_SETMASK and `sigsuspend` separately, a signal that occurs between these functions is often not noticed by `sigsuspend`.

In normal usage, a signal is blocked by using the `sigprocmask` function at the beginning of a critical section. The process then determines whether there is work for it to do. If there is no work, the process waits for work by calling `sigsuspend` with the mask previously returned by `sigprocmask`.

If a signal is caught by the calling process and control is returned from the signal handler, the calling process resumes execution after `sigsuspend`, which always returns a value of `-1` and sets `errno` to `EINTR`.

See also `sigpause` and `sigprocmask` in this section.

---

## sigvec

Permanently assigns a handler for a specific signal.

### Format

```
#include <signal.h>

int sigvec (int sigint, struct sigvec *sv, struct sigvec *osv);
```

### Arguments

**sigint**

The signal identifier.

**sv**

Pointer to a sigvec structure (see the Description section).

**osv**

If *osv* is not NULL, the previous handling information for the signal is returned.

### Description

If *sv* is not NULL, it specifies the address of a structure containing a pointer to a handler routine and mask to be used when delivering the specified signal, and a flag indicating whether the signal is to be processed on an alternative stack. If *sv*→*onstack* has a value of 1, the system delivers the signal to the process on a signal stack specified with sigstack.

The sigvec function establishes a handler that remains established until explicitly removed or until the image terminates.

The sigvec structure is defined in the <signal.h> header file as follows:

```
struct sigvec
{
 int (*handler)();
 int mask;
 int onstack;
};
```

See Section 4.2 for more information on signal handling.

### Return Values

|    |                                    |
|----|------------------------------------|
| 0  | Indicates that the call succeeded. |
| -1 | Indicates that an error occurred.  |

## sin

---

## sin

Returns the sine of its radian argument.

### Format

```
#include <math.h>

double sin (double x);
float sinf (float x); (Alpha only)
long double sinl (long double x); (Alpha only)
double sind (double x); (Alpha only)
float sindf (float x); (Alpha only)
long double sindl (long double x); (Alpha only)
```

### Argument

**x**  
A radian expressed as a floating-point number.

### Description

The `sin` functions compute the sine of  $x$  measured in radians.  
The `sind` functions compute the sine of  $x$  measured in degrees.

### Return Values

|                  |                                                                                                 |
|------------------|-------------------------------------------------------------------------------------------------|
| <code>x</code>   | The sine of the argument.                                                                       |
| <code>NaN</code> | $x = \pm\text{Infinity}$ or <code>NaN</code> ; <code>errno</code> is set to <code>EDOM</code> . |
| <code>0</code>   | Underflow occurred; <code>errno</code> is set to <code>ERANGE</code> .                          |

---

## sinh

Returns the hyperbolic sine of its argument.

### Format

```
#include <math.h>

double sinh (double x);
float sinhf (float x); (Alpha only)
long double sinhl (long double x); (Alpha only)
```

### Argument

**x**  
A real number.

### Return Values

|          |                                             |
|----------|---------------------------------------------|
| n        | The hyperbolic sine of the argument.        |
| HUGE_VAL | Overflow occurred; errno is set to ERANGE.  |
| 0        | Underflow occurred; errno is set to ERANGE. |
| NaN      | <i>x</i> is NaN; errno is set to EDOM.      |

## sleep

---

## sleep

Suspends the execution of the current process for at least the number of seconds indicated by its argument.

### Format

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

```
unsigned int sleep (unsigned seconds); (_DECC_V4_SOURCE)
```

```
int sleep (unsigned seconds); (not _DECC_V4_SOURCE)
```

### Argument

#### **seconds**

The number of seconds.

### Description

The sleep function sleeps for the specified number of seconds, or until a signal is received, or until the process executes a call to SYS\$WAKE.

If a SIGALRM signal is generated, but blocked or ignored, the sleep function returns. For all other signals, a blocked or ignored signal does not cause sleep to return.

### Return Values

|   |                                                                             |
|---|-----------------------------------------------------------------------------|
| x | The number of seconds that the process awoke early.                         |
| 0 | If the process slept the full number of seconds specified by <i>seconds</i> |

---

## sprintf

Performs formatted output to a string in memory.

### Format

```
#include <stdio.h>

int sprintf (char *str, const char *format_spec, ...);
```

### Arguments

#### **str**

The address of the string that will receive the formatted output. It is assumed that this string is large enough to hold the output.

#### **format\_spec**

A pointer to a character string that contains the format specification. For more information about format specifications and conversion characters, see Chapter 2.

...

Optional expressions whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, you may omit the output sources. Otherwise, the function calls must have at least as many output sources as there are conversion specifications, and the conversion specifications must match the types of the output sources.

Conversion specifications are matched to output sources in left-to-right order. Excess output pointers, if any, are ignored.

### Description

A null character is automatically appended to the end of the output string. Consider the following example of a conversion specification:

```
#include <stdio.h>

main()
{
 int temp = 4, temp2 = 17;
 char s[80];

 sprintf(s, "The answers are %d, and %d.", temp, temp2);
}
```

In this example, character string *s* has the following contents:

The answers are 4, and 17.

For a complete description of the format specification and the output source, see Chapter 2.

## **sprintf**

### **Return Value**

x

The number of characters placed in the output string, not including the final null character.

Negative value

Indicates an output error occurred. The function sets `errno`. For a list of `errno` values set by this function, see `fprintf` in this section.



---

## sqrt

Returns the square root of its argument.

### Format

```
#include <math.h>

double sqrt (double x);
float sqrtf (float x); (Alpha only)
long double sqrtl (long double x); (Alpha only)
```

### Argument

**x**  
A real number.

### Return Values

|     |                                                                   |
|-----|-------------------------------------------------------------------|
| val | The square root of $x$ , if $x$ is nonnegative.                   |
| 0   | $x$ is negative; <code>errno</code> is set to <code>EDOM</code> . |
| NaN | $x$ is NaN; <code>errno</code> is set to <code>EDOM</code> .      |

## srand

---

## srand

Initializes the pseudorandom-number generator rand.

### Format

```
#include <math.h>

void srand (unsigned int seed);
```

### Argument

**seed**  
An unsigned integer.

### Description

The `srand` function uses the argument as a seed for a new sequence of pseudorandom numbers to be returned by subsequent calls to `rand`.

If `rand` is called before any calls to `srand`, the sequence of pseudorandom numbers is generated as if the seed were set to 1.

---

## srand48

Initializes a 48-bit random-number generator.

### Format

```
#include <stdlib.h>

void srand48 (long int seed_val);
```

### Argument

#### **seed\_val**

The initialization value to begin randomization. Changing this value changes the randomization pattern.

### Description

The `srand48` function initializes the random-number generator. You can use this function in your program before calling the `drand48`, `lrand48`, or `mrand48` functions. (Although it is not recommended practice, constant default initializer values are supplied automatically if you call `drand48`, `lrand48`, or `mrand48` without calling an initialization function).

The function works by generating a sequence of 48-bit integer values,  $X_i$ , according to the linear congruential formula:

$$X_{n+1} = (aX_n + c) \bmod m \quad n \geq 0$$

The argument  $m$  equals  $2^{48}$ , so 48-bit integer arithmetic is performed. Unless you invoke the `lcong48` function, the multiplier value  $a$  and the addend value  $c$  are:

$$\begin{aligned} a &= 5DEECE66D_{16} = 2736731631558 \\ c &= B_{16} = 138 \end{aligned}$$

The initializer function `srand48` sets the high-order 32 bits of  $X_i$  to the low-order 32 bits contained in its argument. The low-order 16 bits of  $X_i$  are set to the arbitrary value  $330E_{16}$ .

See also `drand48`, `lrand48`, and `mrand48` in this section.

## srandom

---

## srandom

Initializes the pseudorandom-number generator `random`.

### Format

```
int srandom (unsigned seed);
```

### Argument

**seed**

An initial seed value.

### Description

The `srandom` function uses the argument as a seed for a new sequence of pseudorandom numbers to be returned by subsequent calls to `random`. This function has virtually the same calling sequence and initialization properties as the `rand` function, but produce sequences that are more random.

The `srandom` function initializes the current state with the initial seed value. The `srandom` function, unlike the `rand` function, does not return the old seed because the amount of state information used is more than a single word.

See also `rand`, `rand`, `random`, `setstate`, and `initstate` in this section.

### Return Values

|    |                                                                          |
|----|--------------------------------------------------------------------------|
| 0  | Indicates success. Initializes the state seed.                           |
| -1 | Indicates an error, further specified in the global <code>errno</code> . |

---

## sscanf

Reads input from a character string in memory, interpreting it according to the format specification.

### Format

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

```
int sscanf (const char *str, const char *format_spec, ...);
```

### Arguments

#### **str**

The address of the character string that provides the input text to `sscanf`.

#### **format\_spec**

A pointer to a character string that contains the format specification. For more information about format specifications and conversion characters, see Chapter 2.

...

Optional expressions whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, you can omit the input pointers. Otherwise, the function calls must have at least as many input pointers as there are conversion specifications, and the conversion specifications must match the types of the input pointers.

Conversion specifications are matched to input sources in left-to-right order. Excess input pointers, if any, are ignored.

### Description

The following is an example of a conversion specification:

```
main ()
{
 char str[] = "4 17";
 int temp,
 temp2;

 sscanf(str, "%d %d", &temp, &temp2);
 printf("The answers are %d and %d.", temp, temp2);
}
```

This example produces the following output:

**\$ RUN EXAMPLE**

The answers are 4 and 17.

For a complete description of the format specification and the input pointers, see Chapter 2.

## **sscanf**

### **Return Values**

x

The number of successfully matched and assigned input items.

EOF

Indicates that a read error occurred before any conversion. The function sets `errno`. For a list of the values set by this function, see `fscanf` in this section.

---

## ssignal

Allows you to specify the action to take when a particular signal is raised.

### Format

```
#include <signal.h>

void (*ssignal (int sig, void (*func) (int, ...))) (int, ...);
```

### Arguments

**sig**

A number or mnemonic associated with a signal. The symbolic constants for signal values are defined in the `<signal.h>` header file (see Chapter 4).

**func**

The action to take when the signal is raised, or the address of a function that is executed when the signal is raised.

### Description

The `ssignal` function is equivalent to the `signal` function except for the return value on error conditions.

Since the `signal` function is defined by the ANSI C standard and the `ssignal` function is not, use `signal` for greater portability.

See Section 4.2 for more information on signal handling.

### Return Values

|   |                                                                                                                                                                                          |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x | The address of the function previously established as the action for the signal. The address may be the value <code>SIG_DFL (0)</code> or <code>SIG_IGN (1)</code> .                     |
| 0 | Indicates errors. For this reason, there is no way to know whether a return status of 0 indicates failure, or whether it indicates that a previous action was <code>SIG_DFL (0)</code> . |

## [w]standend

---

## [w]standend

Deactivate the boldface attribute for the specified window. The `standend` function operates on the `stdscr` window.

### Format

```
#include <curses.h>

int standend (void);

int wstandend (WINDOW *win);
```

### Argument

**win**  
A pointer to the window.

### Description

The `standend` and `wstandend` functions are equivalent to `clrattr` and `wclrattr` called with the attribute `_BOLD`.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |



---

## **[w]standout**

Activate the boldface attribute of the specified window. The `standout` function acts on the `stdscr` window.

### **Format**

```
#include <curses.h>

int standout (void);

int wstandout (WINDOW *win);
```

### **Argument**

**win**  
A pointer to the window.

### **Description**

The `standout` and `wstandout` functions are equivalent to `setattr` and `wsetattr` called with the attribute `_BOLD`.

### **Return Values**

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

## stat

---

## stat

Accesses information about the specified file.

### Format

```
#include <stat.h>
```

```
int stat (const char *file_spec, struct stat *buffer); (ISO POSIX-1)
```

```
int stat (const char *file_spec, struct stat *buffer, . . .); (COMPAQ C Extension)
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `stat` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Arguments

#### **file\_spec**

A valid OpenVMS or UNIX style file specification (no wildcards). Read, write, or execute permission of the named file is not required, but you must be able to reach all directories listed in the file specification leading to the file. For more information about UNIX style file specifications, see Chapter 1.

#### **buffer**

A pointer to a structure of type `stat_t` that is defined in the `<stat.h>` header file.

The argument receives information about the particular file. The members of the structure pointed to by *buffer* are described as follows:

| Member                  | Type                  | Definition                                                   |
|-------------------------|-----------------------|--------------------------------------------------------------|
| <code>st_dev</code>     | <code>dev_t</code>    | Pointer to the physical device name                          |
| <code>st_ino[3]</code>  | <code>ino_t</code>    | Three words to receive the file ID                           |
| <code>st_mode</code>    | <code>mode_t</code>   | File “mode” ( <code>prot</code> , <code>dir</code> , . . . ) |
| <code>st_nlink</code>   | <code>nlink_t</code>  | For UNIX system compatibility only                           |
| <code>st_uid</code>     | <code>uid_t</code>    | Owner user ID                                                |
| <code>st_gid</code>     | <code>gid_t</code>    | Group member: from <code>st_uid</code>                       |
| <code>st_rdev</code>    | <code>dev_t</code>    | UNIX system compatibility – always 0                         |
| <code>st_size</code>    | <code>off_t</code>    | File size, in bytes                                          |
| <code>st_atime</code>   | <code>time_t</code>   | File access time; always the same as <code>st_mtime</code>   |
| <code>st_mtime</code>   | <code>time_t</code>   | Last modification time                                       |
| <code>st_ctime</code>   | <code>time_t</code>   | File creation time                                           |
| <code>st_fab_rfm</code> | <code>char</code>     | Record format                                                |
| <code>st_fab_rat</code> | <code>char</code>     | Record attributes                                            |
| <code>st_fab_fsz</code> | <code>char</code>     | Fixed header size                                            |
| <code>st_fab_mrs</code> | <code>unsigned</code> | Record size                                                  |

The types `dev_t`, `ino_t`, `off_t`, `mode_t`, `nlink_t`, `uid_t`, `gid_t`, and `time_t`, are defined in the `<stat.h>` header file. However, when compiling for compatibility (`/DEFINE=_DECC_V4_SOURCE`), only `dev_t`, `ino_t`, and `off_t` are defined.

The `off_t` data type is either a 32-bit integer or 64-bit integer. The 64-bit interface allows for file sizes greater than 2 gigabytes, and can be selected at compile time by defining the `_LARGEFILE` feature-test macro as follows:

```
CC/DEFINE=_LARGEFILE
```

As of OpenVMS Version 7.0, times are given in seconds since the Epoch (00:00:00 GMT, January 1, 1970).

The `st_mode` structure member is the status information mode defined in the `<stat.h>` header file. The `st_mode` bits are described as follows:

| Bits    | Constant | Definition                       |
|---------|----------|----------------------------------|
| 0170000 | S_IFMT   | Type of file                     |
| 0040000 | S_IFDIR  | Directory                        |
| 0020000 | S_IFCHR  | Character special                |
| 0060000 | S_IFBLK  | Block special                    |
| 0100000 | S_IFREG  | Regular                          |
| 0030000 | S_IFMPC  | Multiplexed char special         |
| 0070000 | S_IFMPB  | Multiplexed block special        |
| 0004000 | S_ISUID  | Set user ID on execution         |
| 0002000 | S_ISGID  | Set group ID on execution        |
| 0001000 | S_ISVTX  | Save swapped text even after use |
| 0000400 | S_IREAD  | Read permission, owner           |
| 0000200 | S_IWRITE | Write permission, owner          |
| 0000100 | S_IEXEC  | Execute/search permission, owner |

...

An optional default file-name string.

This is the only optional RMS keyword that can be specified for the `stat` function. See the description of the `creat` function for the full list of optional RMS keywords and their values.

## Description

The `stat` function does not work on remote network files.

If the file is a record file, the `st_size` field includes carriage-control information. Consequently, the `st_size` value will not correspond to the number of characters that can be read from the file.

The physical device name string referred to by the `st_dev` member of the `stat` structure is overwritten by the next `stat` call.

### Note (Alpha only)

On OpenVMS Alpha systems, the `stat`, `fstat`, `utime`, and `utimes` functions have been enhanced to take advantage of the new file-system support for POSIX-compliant file timestamps.

## stat

This support is available only on ODS-5 devices on OpenVMS Alpha systems beginning with a version of OpenVMS Alpha after Version 7.3.

Before this change, the `stat` and `fstat` functions were setting the values of the `st_ctime`, `st_mtime`, and `st_atime` fields based on the following file attributes:

- `st_ctime` - `ATR$C_CREDATE` (file creation time)
- `st_mtime` - `ATR$C_REVDATE` (file revision time)
- `st_atime` - was always set to `st_mtime` because no support for file access time was available

Also, for the file-modification time, `utime` and `utimes` were modifying the `ATR$C_REVDATE` file attribute, and ignoring the file-access-time argument.

After the change, for a file on an ODS-5 device, the `stat` and `fstat` functions set the values of the `st_ctime`, `st_mtime`, and `st_atime` fields based on the following new file attributes:

- `st_ctime` - `ATR$C_ATTDATE` (last attribute modification time)
- `st_mtime` - `ATR$C_MODDATE` (last data modification time)
- `st_atime` - `ATR$C_ACCDATE` (last access time)

If `ATR$C_ACCDATE` is zero, as on an ODS-2 device, the `stat` and `fstat` functions set `st_atime` to `st_mtime`.

For the file-modification time, the `utime` and `utimes` functions modify both the `ATR$C_REVDATE` and `ATR$C_MODDATE` file attributes. For the file-access time, these functions modify the `ATR$C_ACCDATE` file attribute. Setting the `ATR$C_MODDATE` and `ATR$C_ACCDATE` file attributes on an ODS-2 device has no effect.

For compatibility, the old behavior of `stat`, `fstat`, `utime`, and `utimes` remains the default, regardless of the kind of device.

The new behavior must be explicitly enabled by defining the `DECC$EFS_FILE_TIMESTAMPS` logical name to "ENABLE" before invoking the application. Setting this logical does not affect the behavior of `stat`, `fstat`, `utime`, and `utimes` for files on an ODS-2 device.

---

## Return Values

|    |                                                                                                       |
|----|-------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                    |
| -1 | Indicates an error other than a privilege violation; <code>errno</code> is set to indicate the error. |
| -2 | Indicates a privilege violation.                                                                      |

---

## strcasecmp

Does a case-insensitive comparison of two 7-bit ASCII strings.

### Format

```
#include <strings.h>

int strcasecmp (const char *s1, const char *s2);
```

### Arguments

**s1**  
The first of two strings to compare.

**s2**  
The second of two strings to compare.

### Description

The `strcasecmp` function is case-insensitive. The returned lexicographic difference reflects a conversion to lowercase.

The `strcasecmp` function works for 7-bit ASCII compares only. Do not use this function for internationalized applications.

### Return Value

|   |                                                                                                                                                                         |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n | An integer value greater than, equal to, or less than 0 (zero), depending on whether the <i>s1</i> string is greater than, equal to, or less than the <i>s2</i> string. |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## strcat

---

## strcat

Concatenates *str\_2*, including the terminating null character, to the end of *str\_1*.

### Format

```
#include <string.h>

char *strcat (char *str_1, const char *str_2);
```

### Function Variants

The `strcat` function has variants named `_strcat32` and `_strcat64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**str\_1, str\_2**  
Pointers to null-terminated character strings.

### Description

See also `strncat` in this section.

### Return Value

|   |                                                                                                                        |
|---|------------------------------------------------------------------------------------------------------------------------|
| x | The address of the first argument, <i>str_1</i> , which is assumed to be large enough to hold the concatenated result. |
|---|------------------------------------------------------------------------------------------------------------------------|

### Example

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

/* This program concatenates two strings using the strcat */
/* function, and then manually compares the result of strcat */
/* to the expected result. */

#define S1LENGTH 10
#define S2LENGTH 8

main()
{
 static char s1buf[S1LENGTH + S2LENGTH] = "abcmnxyz";
 static char s2buf[] = " orthis";
 static char test1[] = "abcmnxyz orthis";

 int i;
 char *status;

 /* Take static buffer s1buf, concatenate static buffer */
 /* s2buf to it, and compare the answer in s1buf with the */
 /* static answer in test1. */

 status = strcat(s1buf, s2buf);
 for (i = 0; i <= S1LENGTH + S2LENGTH - 2; i++) {
 /* Check for correct returned string. */

```

```
 if (test1[i] != slbuf[i])
 printf("error in strcat");
 }
}
```

## strchr

---

## strchr

Returns the address of the first occurrence of a given character in a null-terminated string. The terminating null character is considered to be part of the string.

### Format

```
#include <string.h>

char *strchr (const char *str, int character);
```

### Function Variants

The `strchr` function has variants named `_strchr32` and `_strchr64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**str**  
A pointer to a null-terminated character string.

**character**  
An object of type `int`.

### Description

See also `strrchr` in this section.

### Return Values

|                   |                                                                 |
|-------------------|-----------------------------------------------------------------|
| <code>x</code>    | The address of the first occurrence of the specified character. |
| <code>NULL</code> | Indicates that the character does not occur in the string.      |

### Example

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

main()
{
 static char s1buf[] = "abcdefghijkl lkjihgfedcba";
 int i;
 char *status;

 /* This program checks the strchr function by incrementally */
 /* going through a string that ascends to the middle and then */
 /* descends towards the end. */
 for (i = 0; s1buf[i] != '\0' && s1buf[i] != ' '; i++) {
 status = strchr(s1buf, s1buf[i]);
 /* Check for pointer to leftmost character - test 1. */
 }
```



```
 if (status != &slbuf[i])
 printf("error in strchr");
 }
}
```

## strcmp

---

## strcmp

Compares two ASCII character strings and returns a negative, 0, or positive integer, indicating that the ASCII values of the individual characters in the first string are less than, equal to, or greater than the values in the second string.

### Format

```
#include <string.h>

int strcmp (const char *str_1, const char *str_2);
```

### Arguments

**str\_1, str\_2**  
Pointers to character strings.

### Description

The strings are compared until a null character is encountered or until the strings differ.

### Return Values

|     |                                                            |
|-----|------------------------------------------------------------|
| < 0 | Indicates that <i>str_1</i> is less than <i>str_2</i> .    |
| = 0 | Indicates that <i>str_1</i> equals <i>str_2</i> .          |
| > 0 | Indicates that <i>str_1</i> is greater than <i>str_2</i> . |

---

## strcoll

Compares two strings and returns an integer that indicates if the strings differ and how they differ. The function uses the collating information in the LC\_COLLATE category of the current locale to determine how the comparison is performed.

### Format

```
#include <string.h>

int strcoll (const char *s1, const char *s2);
```

### Arguments

**s1, s2**  
Pointers to character strings.

### Description

The strcoll function, unlike strcmp, compares two strings in a locale-dependent manner. Because no value is reserved for error indication, the application must check for one by setting errno to 0 before the function call and testing it after the call.

See also the strxfrm function in this section.

### Return Values

|     |                                                      |
|-----|------------------------------------------------------|
| < 0 | Indicates that <i>s1</i> is less than <i>s2</i> .    |
| = 0 | Indicates that the strings are equal.                |
| > 0 | Indicates that <i>s1</i> is greater than <i>s2</i> . |

## strcpy

---

## strcpy

Copies all of *source*, including the terminating null character, into *dest*.

### Format

```
#include <string.h>

char *strcpy (char *dest, const char *source);
```

### Function Variants

The `strcpy` function has variants named `_strcpy32` and `_strcpy64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**  
Pointer to the destination character string.

**source**  
Pointer to the source character string.

### Description

The `strcpy` function copies *source* into *dest*, and stops after copying *source*'s null character.

The behavior of this function is undefined if the area pointed to by *dest* overlaps the area pointed to by *source*.

### Return Value

|   |                              |
|---|------------------------------|
| x | The address of <i>dest</i> . |
|---|------------------------------|

---

## strcspn

Returns the length of the prefix of a string that consists entirely of characters not in a specified set of characters.

### Format

```
#include <string.h>

size_t strcspn (const char *str, const char *charset);
```

### Arguments

**str**

A pointer to a character string. If this character string is a null string, 0 is returned.

**charset**

A pointer to a character string containing the set of characters.

### Description

The `strcspn` function scans the characters in the string, stops when it encounters a character found in *charset*, and returns the length of the string's initial segment formed by characters not found in *charset*.

If none of the characters match in the character strings pointed to by *str* and *charset*, `strcspn` returns the length of string.

### Return Value

|   |                            |
|---|----------------------------|
| x | The length of the segment. |
|---|----------------------------|

## strdup

---

## strdup

Finds and points to a duplicate string.

### Format

```
#include <string.h>

char *strdup (const char *s1);
```

### Function Variants

The `strdup` function has variants named `_strdup32` and `_strdup64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Argument

**s1**  
The first of two strings to compare.

### Description

The `strdup` function returns a pointer to a string that is an exact duplicate of the string pointed to by *s1*. The `malloc` function is used to allocate space for the new string. The *strdup* function is provided for compatibility with existing systems.

### Return Values

|      |                                    |
|------|------------------------------------|
| x    | A pointer to the resulting string. |
| NULL | Indicates an error.                |

---

## strerror

Maps the error number in *error\_code* to a locale-dependent error message string.

### Format

```
#include <string.h>

char *strerror (int error_code); (ANSI C)

char *strerror (int error_code, int vms_error_code); (COMPAQ C Extension)
```

### Arguments

#### **error\_code**

An error code.

#### **vms\_error\_code**

An OpenVMS error code.

### Description

The `strerror` function uses the error number in *error\_code* to retrieve the appropriate locale-dependent error message. The contents of the error message strings are determined by the LC\_MESSAGES category of the program's current locale.

When a program is not compiled with any standards-related feature-test macros (see Section 1.5.1), `strerror` has a second argument (*vms\_error\_code*), which is used in the following way:

- If *error\_code* is `EVMSError` and there is a second argument, then that second argument is used as the `vaxc$errno` value.
- If *error\_code* is `EVMSError` and there is no second argument, look at `vaxc$errno` to get the OpenVMS error condition.

See the `strerror` example.

Use of the second argument is not included in the ANSI C definition of `strerror` and is, therefore, not portable.

Because no return value is reserved to indicate an error, applications should set the value of `errno` to 0, call `strerror`, and then test the value of `errno`; a nonzero value indicates an error condition.

### Return Values

x

A pointer to a buffer containing the appropriate error message. Do not modify this buffer in your programs. Moreover, calls to the `strerror` function may overwrite this buffer with a new message.

## strerror

### Example

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <ssdef.h>

main()
{
 puts(strerror(EVMSERR));
 errno = EVMSERR;
 vaxc$errno = SS$_LINKEEXIT;
 puts(strerror(errno));
 puts(strerror(EVMSERR, SS$_ABORT));
 exit(1);
}
```

Running this example produces the following output:

```
nontranslatable vms error code: <none>
network partner exited
abort
```



---

## strfmon

Converts a number of monetary values into a string. The conversion is controlled by a format string.

### Format

```
#include <monetary.h>

ssize_t strfmon (char *s, size_t maxsize, const char *format, ...);
```

### Arguments

**s**

A pointer to the resultant string.

**maxsize**

The maximum number of bytes to be stored in the resultant string.

**format**

A pointer to a string that controls the format of the output string.

...

The monetary values of type double that are to be formatted for the output string. There should be as many values as there are conversion specifications in the format string pointed to by *format*. The function fails if there are insufficient values. Excess arguments are ignored.

### Description

The `strfmon` function creates a string pointed to by *s*, using the monetary values supplied. A maximum of *maxsize* bytes is copied to *s*.

The format string pointed to by *format* consists of ordinary characters and conversion specifications. All ordinary characters are copied unchanged to the output string. A conversion specification defines how one of the monetary values supplied is formatted in the output string.

A conversion specification consists of a percent character (%), followed by a number of optional characters (see Table REF-5), and concluding with a conversion specifier (see Table REF-6).

If any of the optional characters listed in Table REF-5 is included in a conversion specification, they must appear in the order shown.

Table REF–5 Optional Characters in strfmon Conversion Specifications

| Character         | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>=character</i> | Use <i>character</i> as the numeric fill character if a left precision is specified. The default numeric fill character is the space character. The fill character must be representable as a single byte in order to work with precision and width count. This conversion specifier is ignored unless a left precision is specified, and it does not affect width filling, which always uses the space character.                                                                                                           |
| ^                 | Do not use separator characters to format the number. By default, the digits are grouped according to the <i>mon_grouping</i> field in the LC_MONETARY category of the current locale.                                                                                                                                                                                                                                                                                                                                       |
| +                 | Add the string specified by the <i>positive_sign</i> or <i>negative_sign</i> fields in the current locale. If <i>p_sign_posn</i> or <i>n_sign_posn</i> is set to zero, then parentheses are used by default to indicate negative values. Otherwise, sign strings are used to indicate the sign of the value. You cannot use a + and a ( in the same conversion specification.                                                                                                                                                |
| (                 | Enclose negative values within parentheses. The default is taken from the <i>p_sign_posn</i> and <i>n_sign_posn</i> fields in the current locale. If <i>p_sign_posn</i> or <i>n_sign_posn</i> is set to zero, then parentheses are used by default to indicate negative values. Otherwise, sign strings are used to indicate the sign of the value. You cannot use a + and ( in the same conversion specification.                                                                                                           |
| !                 | Suppress the currency symbol. By default, the currency symbol is included.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| –                 | Left-justify the value within the field. By default, values are right-justified.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| field width       | A decimal integer that specifies the minimum field width in which to align the result of the conversion. The default field width is the smallest field that can contain the result.                                                                                                                                                                                                                                                                                                                                          |
| #left_precision   | A # followed by a decimal integer specifies the number of digits to the left of the radix character. Extra positions are filled by the fill character. By default the precision is the smallest required for the argument. If grouping is not suppressed with the ^ conversion specifier, and if grouping is defined for the current locale, grouping separators are inserted before any fill characters are added. Grouping separators are not applied to fill characters even if the fill character is defined as a digit. |

(continued on next page)

Table REF–5 (Cont.) Optional Characters in strfmon Conversion Specifications

| Character        | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .right_precision | A period (.) followed by a decimal integer specifies the number of digits to the right of the radix character. Extra positions are filled with zeros. The amount is rounded to this number of decimal places. If the right precision is zero, the radix character is not included in the output. By default the right precision is defined by the <i>frac_digits</i> or <i>int_frac_digits</i> field of the current locale. |

Table REF–6 strfmon Conversion Specifiers

| Specifier | Meaning                                                                                                                                                      |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| i         | Use the international currency symbol defined by the <i>int_currency_symbol</i> field in the current locale, unless the currency symbol has been suppressed. |
| n         | Use the local currency symbol defined by the <i>currency_symbol</i> field in the current locale, unless the currency symbol has been suppressed.             |
| %         | Output a % character. The conversion specification must be %%; none of the optional characters is valid with this specifier.                                 |

## Return Values

|    |                                                                                                                                                                                                                                                                                                    |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | The number of bytes written to the string pointed to by s, not including the null-terminating character.                                                                                                                                                                                           |
| –1 | Indicates an error. The function sets errno to one of the following values: <ul style="list-style-type: none"> <li>EINVAL – A conversion specification is syntactically incorrect.</li> <li>E2BIG – Processing the complete format string would produce more than <i>maxsize</i> bytes.</li> </ul> |

## Example

```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>
#include <monetary.h>
#include <errno.h>

#define MAX_BUF_SIZE 124

main()
{
 size_t ret;
 char _buffer[MAX_BUF_SIZE];
 double amount = 102593421;

 /* Display a monetary amount using the en_US.ISO8859-1 */
 /* locale and a range of different display formats. */
}
```

## strfmon

```
if (setlocale(LC_ALL, "en_US.ISO8859-1") == (char *) NULL) {
 perror("setlocale");
 exit(EXIT_FAILURE);
}
ret = strfmon(buffer, MAX_BUF_SIZE, "International: %i\n", amount);
printf(buffer);

ret = strfmon(buffer, MAX_BUF_SIZE, "National: %n\n", amount);
printf(buffer);

ret = strfmon(buffer, MAX_BUF_SIZE, "National: %=*#10n\n", amount);
printf(buffer);

ret = strfmon(buffer, MAX_BUF_SIZE, "National: %(n\n", -1 * amount);
printf(buffer);

ret = strfmon(buffer, MAX_BUF_SIZE, "National: %^!n\n", amount);
printf(buffer);
}
```

Running the example program produces the following result:

```
International: USD 102,593,421.00
National: $102,593,421.00
National: $**102,593,421.00
National: ($102,593,421.00)
National: 102593421.00
```

---

## strftime

Uses date and time information stored in a `tm` structure, to create an output string. The format of the output string is controlled by a format string.

### Format

```
#include <time.h>
```

```
size_t strftime (char *s, size_t maxsize, const char *format, const struct tm *timeptr);
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `strftime` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Arguments

**s**

A pointer to the resultant string.

**maxsize**

The maximum number of bytes to be stored in the resultant string, including the null terminator.

**format**

A pointer to a string that controls the format of the output string.

**timeptr**

A pointer to the local time (`tm`) structure. The `tm` structure is defined in the `<time.h>` header file.

### Description

The `strftime` function uses data in the structure pointed to by *timeptr* to create the string pointed to by *s*. A maximum of *maxsize* bytes is copied to *s*.

The format string consists of zero or more conversion specifications and ordinary characters. All ordinary characters (including the terminating null character) are copied unchanged into the output string. A conversion specification defines how data in the `tm` structure is formatted in the output string.

A conversion specification consists of a percent (%) character followed by one or more optional characters (see Table REF-7), and concluding with a conversion specifier (see Table REF-8). If any of the optional characters listed in Table REF-7 are specified, they must appear in the order shown in the table.

The `strftime` function behaves as if it called `tzset`.

**Table REF–7 Optional Elements of strptime Conversion Specifications**

| Element     | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| –           | Optional with the field width to specify that the field is left-justified and padded with spaces. This cannot be used with the 0 element.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 0           | Optional with the field width to specify that the field is right-justified and padded with zeros. This cannot be used with the – element.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| field width | A decimal integer that specifies the maximum field width                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| .precision  | <p>A decimal integer that specifies the precision of data in a field.</p> <p>For the d, H, I, j, m, M, o, S, U, w, W, y, and Y conversion specifiers, the precision specifier is the minimum number of digits to appear in the field. If the conversion specification has fewer digits than that specified by the precision, leading zeros are added.</p> <p>For the a, A, b, B, c, D, E, h, n, N, p, r, t, T, x, X, Z, and % conversion specifiers, the precision specifier is the maximum number of characters to appear in the field. If the conversion specification has more characters than that specified by the the precision, characters are truncated on the right.</p> <p>The default precision for the d, H, I, m, M, o, S, U, w, W, y and Y conversion specifiers is 2; the default precision for the j conversion specifier is 3.</p> |

Note that the list of conversion specifications in Table REF–7 are extensions to the XPG4 specification.

Table REF–8 lists the conversion specifiers. The `strptime` function uses fields in the `LC_TIME` category of the program's current locale to provide a value. For example, if `%B` is specified, the function accesses the *mon* field in `LC_TIME` to find the full month name for the month specified in the `tm` structure. The result of using invalid conversion specifiers is undefined.

**Table REF–8 strptime Conversion Specifiers**

| Specifier | Replaced by                                                                                                |
|-----------|------------------------------------------------------------------------------------------------------------|
| a         | The locale's abbreviated weekday name                                                                      |
| A         | The locale's full weekday name                                                                             |
| b         | The locale's abbreviated month name                                                                        |
| B         | The locale's full month name                                                                               |
| c         | The locale's appropriate date and time representation                                                      |
| C         | The century number (the year divided by 100 and truncated to an integer) as a decimal number (00 – 99)     |
| d         | The day of the month as a decimal number (01 – 31)                                                         |
| D         | Same as <code>%m/%d/%y</code>                                                                              |
| e         | The day of the month as a decimal number (1 – 31) in a 2 digit field with the leading space character fill |
| Ec        | The locale's alternative date and time representation                                                      |

(continued on next page)

Table REF–8 (Cont.) strptime Conversion Specifiers

| Specifier | Replaced by                                                                                                                                                                                                                                                                                                                                   |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EC        | The name of the base year (period) in the locale's alternative representation                                                                                                                                                                                                                                                                 |
| Ex        | The locale's alternative date representation                                                                                                                                                                                                                                                                                                  |
| EX        | The locale's alternative time representation                                                                                                                                                                                                                                                                                                  |
| Ey        | The offset from the base year (%EC) in the locale's alternative representation                                                                                                                                                                                                                                                                |
| EY        | The locale's full alternative year representation                                                                                                                                                                                                                                                                                             |
| h         | Same as %b                                                                                                                                                                                                                                                                                                                                    |
| H         | The hour (24-hour clock) as a decimal number (00 – 23)                                                                                                                                                                                                                                                                                        |
| I         | The hour (12-hour clock) as a decimal number (01 – 12)                                                                                                                                                                                                                                                                                        |
| j         | The day of the year as a decimal number (001 – 366)                                                                                                                                                                                                                                                                                           |
| m         | The month as a decimal number (01 – 12)                                                                                                                                                                                                                                                                                                       |
| M         | The minute as a decimal number (00 – 59)                                                                                                                                                                                                                                                                                                      |
| n         | The new-line character                                                                                                                                                                                                                                                                                                                        |
| Od        | The day of the month using the locale's alternative numeric symbols                                                                                                                                                                                                                                                                           |
| Oe        | The date of the month using the locale's alternative numeric symbols                                                                                                                                                                                                                                                                          |
| OH        | The hour (24-hour clock) using the locale's alternative numeric symbols                                                                                                                                                                                                                                                                       |
| OI        | The hour (12-hour clock) using the locale's alternative numeric symbols                                                                                                                                                                                                                                                                       |
| Om        | The month using the locale's alternative numeric symbols                                                                                                                                                                                                                                                                                      |
| OM        | The minutes using the locale's alternative numeric symbols                                                                                                                                                                                                                                                                                    |
| OS        | The seconds using the locale's alternative numeric symbols                                                                                                                                                                                                                                                                                    |
| Ou        | The weekday as a number in the locale's alternative representation (Monday=1)                                                                                                                                                                                                                                                                 |
| OU        | The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols                                                                                                                                                                                                                              |
| OV        | The week number of the year (Monday as the first day of the week) as a decimal number (01 – 53) using the locale's alternative numeric symbols. If the week containing January 1 has four or more days in the new year, it is considered as week 1. Otherwise, it is considered as week 53 of the previous year, and the next week is week 1. |
| Ow        | The weekday as a number (Sunday=0) using the locale's alternative numeric symbols                                                                                                                                                                                                                                                             |
| OW        | The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols                                                                                                                                                                                                                              |
| Oy        | The year without the century using the locale's alternative numeric symbols                                                                                                                                                                                                                                                                   |

(continued on next page)

Table REF–8 (Cont.) **strptime Conversion Specifiers**

| Specifier | Replaced by                                                                                                                                                                                                                                                                                    |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| p         | The locale's equivalent of the AM/PM designations associated with a 12-hour clock                                                                                                                                                                                                              |
| r         | The time in AM/PM notation                                                                                                                                                                                                                                                                     |
| R         | The time in 24-hour notation (%H:%M)                                                                                                                                                                                                                                                           |
| S         | The second as a decimal number (00 – 61)                                                                                                                                                                                                                                                       |
| t         | The tab character                                                                                                                                                                                                                                                                              |
| T         | The time (%H:%M:%S)                                                                                                                                                                                                                                                                            |
| u         | The weekday as a decimal number between 1 and 7 (Monday=1)                                                                                                                                                                                                                                     |
| U         | The week number of the year (the first Sunday as the first day of week 1) as a decimal number (00 – 53)                                                                                                                                                                                        |
| V         | The week number of the year (Monday as the first day of the week) as a decimal number (00 – 53). If the week containing January 1 has four or more days in the new year, it is considered as week 1. Otherwise, it is considered as week 53 of the previous year, and the next week is week 1. |
| w         | The weekday as a decimal number (0 [Sunday] – 6)                                                                                                                                                                                                                                               |
| W         | The week number of the year (the first Monday as the first day of week 1) as a decimal number (00 – 53)                                                                                                                                                                                        |
| x         | The locale's appropriate date representation                                                                                                                                                                                                                                                   |
| X         | The locale's appropriate time representation                                                                                                                                                                                                                                                   |
| y         | The year without century as a decimal number (00 – 99)                                                                                                                                                                                                                                         |
| Y         | The year with century as a decimal number                                                                                                                                                                                                                                                      |
| Z         | Time-zone name or abbreviation. If time-zone information is not available, no character is output.                                                                                                                                                                                             |
| %         | Literal % character.                                                                                                                                                                                                                                                                           |

## Return Values

|   |                                                                                                               |
|---|---------------------------------------------------------------------------------------------------------------|
| x | The number of characters placed into the array pointed to by s, not including the terminating null character. |
| 0 | Indicates an error occurred. The contents of the array are indeterminate.                                     |

## Example

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <locale.h>
#include <errno.h>

#define NUM_OF_DATES 7
#define BUF_SIZE 256

/* This program formats a number of different dates, once */
/* using the C locale and then using the fr_FR.ISO8859-1 */
/* locale. Date and time formatting is done using strftime(). */
```



```

main()
{
 int count,
 i;
 char buffer[BUF_SIZE];
 struct tm *tm_ptr;
 time_t time_list[NUM_OF_DATES] =
 {500, 68200000, 694223999, 694224000,
 704900000, 705000000, 705900000};

 /* Display dates using the C locale */
 printf("\nUsing the C locale:\n\n");

 setlocale(LC_ALL, "C");

 for (i = 0; i < NUM_OF_DATES; i++) {
 /* Convert to a tm structure */
 tm_ptr = localtime(&time_list[i]);

 /* Format the date and time */
 count = strftime(buffer, BUF_SIZE,
 "Date: %A %d %B %Y\nTime: %T\n\n", tm_ptr);
 if (count == 0) {
 perror("strftime");
 exit(EXIT_FAILURE);
 }

 /* Print the result */
 printf(buffer);
 }

 /* Display dates using the fr_FR.ISO8859-1 locale */
 printf("\nUsing the fr_FR.ISO8859-1 locale:\n\n");

 setlocale(LC_ALL, "fr_FR.ISO8859-1");

 for (i = 0; i < NUM_OF_DATES; i++) {
 /* Convert to a tm structure */
 tm_ptr = localtime(&time_list[i]);

 /* Format the date and time */
 count = strftime(buffer, BUF_SIZE,
 "Date: %A %d %B %Y\nTime: %T\n\n", tm_ptr);
 if (count == 0) {
 perror("strftime");
 exit(EXIT_FAILURE);
 }

 /* Print the result */
 printf(buffer);
 }
}

```

Running the example program produces the following result:

Using the C locale:

Date: Thursday 01 January 1970  
Time: 00:08:20

Date: Tuesday 29 February 1972  
Time: 08:26:40

Date: Tuesday 31 December 1991  
Time: 23:59:59

Date: Wednesday 01 January 1992  
Time: 00:00:00

Date: Sunday 03 May 1992  
Time: 13:33:20

## strftime

Date: Monday 04 May 1992  
Time: 17:20:00

Date: Friday 15 May 1992  
Time: 03:20:00

Using the fr\_FR.ISO8859-1 locale:

Date: jeudi 01 janvier 1970  
Time: 00:08:20

Date: mardi 29 février 1972  
Time: 08:26:40

Date: mardi 31 décembre 1991  
Time: 23:59:59

Date: mercredi 01 janvier 1992  
Time: 00:00:00

Date: dimanche 03 mai 1992  
Time: 13:33:20

Date: lundi 04 mai 1992  
Time: 17:20:00

Date: vendredi 15 mai 1992  
Time: 03:20:00

---

## strlen

Returns the length of a string of ASCII characters. The returned length does not include the terminating null character (`\0`).

### Format

```
#include <string.h>

size_t strlen (const char *str);
```

### Argument

**str**  
A pointer to the character string.

### Return Value

|   |                           |
|---|---------------------------|
| x | The length of the string. |
|---|---------------------------|

## strncasecmp

---

## strncasecmp

Does a case-insensitive comparison between two 7-bit ASCII strings.

### Format

```
#include <strings.h>
```

```
int strncasecmp (const char *s1, const char *s2, size_t n);
```

### Arguments

**s1**

The first of two strings to compare.

**s2**

The second of two strings to compare.

**n**

The maximum number of bytes in a string to compare.

### Description

The `strncasecmp` function is case-insensitive. The returned lexicographic difference reflects a conversion to lowercase. The `strncasecmp` function is similar to the `strcasecmp` function, but also compares size. If the size specified by *n* is read before a NULL, the comparison stops.

The `strcasecmp` function works for 7-bit ASCII compares only. Do not use this function for internationalized applications.

### Return Value

*n*

An integer value greater than, equal to, or less than 0 (zero), depending on whether *s1* is greater than, equal to, or less than *s2*.

---

## strncat

Appends not more than *maxchar* characters from *str\_2* to the end of *str\_1*.

### Format

```
#include <string.h>

char *strncat (char *str_1, const char *str_2, size_t maxchar);
```

### Function Variants

The `strncat` function has variants named `_strncat32` and `_strncat64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**str\_1, str\_2**

Pointers to null-terminated character strings.

**maxchar**

The number of characters to concatenate from *str\_2*, unless `strncat` first encounters a null terminator in *str\_2*. If *maxchar* is 0, no characters are copied from *str\_2*.

### Description

A null character is always appended to the result of the `strncat` function. If `strncat` reaches the specified maximum, it sets the next byte in *str\_1* to the null character.

### Return Value

|   |                                                                                                                        |
|---|------------------------------------------------------------------------------------------------------------------------|
| x | The address of the first argument, <i>str_1</i> , which is assumed to be large enough to hold the concatenated result. |
|---|------------------------------------------------------------------------------------------------------------------------|

## strncmp

---

## strncmp

Compares not more than *maxchar* characters of two ASCII character strings and returns a negative, 0, or positive integer, indicating that the ASCII values of the individual characters in the first string are less than, equal to, or greater than the values in the second string.

### Format

```
#include <string.h>
```

```
int strncmp (const char *str_1, const char *str_2, size_t maxchar);
```

### Arguments

**str\_1, str\_2**

Pointers to character strings.

**maxchar**

The maximum number of characters (beginning with the first) to search in both *str\_1* and *str\_2*. If *maxchar* is 0, no comparison is performed and 0 is returned (the strings are considered equal).

### Description

The `strncmp` function compares no more than *maxchar* characters from the string pointed to by *str\_1* to the string pointed to by *str\_2*. The strings are compared until a null character is encountered, the strings differ, or *maxchar* is reached. Characters that follow a difference or a null character are not compared.

### Return Values

|     |                                                            |
|-----|------------------------------------------------------------|
| < 0 | Indicates that <i>str_1</i> is less than <i>str_2</i> .    |
| = 0 | Indicates that <i>str_1</i> equals <i>str_2</i> .          |
| > 0 | Indicates that <i>str_1</i> is greater than <i>str_2</i> . |

### Examples

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

 main()
 {
 printf("%d\n", strncmp("abcde", "abc", 3));
 }
```

When linked and executed, this example returns 0, because the first 3 characters of the 2 strings are equal:

```
$ run tmp
0
```

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

 main()
 {
 printf("%d\n", strncmp("abcde", "abc", 4));
 }
```

When linked and executed, this example returns a value greater than 0 because the first 4 characters of the 2 strings are not equal (The "d" in the first string is not equal to the null character in the second):

```
$ run tmp
 100
```

## strncpy

---

## strncpy

Copies not more than *maxchar* characters from *source* into *dest*.

### Format

```
#include <string.h>
```

```
char *strncpy (char *dest, const char *source, size_t maxchar);
```

### Function Variants

The `strncpy` function has variants named `_strncpy32` and `_strncpy64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**

Pointer to the destination character string.

**source**

Pointer to the source character string.

**maxchar**

The maximum number of characters to copy from *source* to *dest* up to but not including the null terminator of *source*.

### Description

The `strncpy` function copies no more than *maxchar* characters from *source* to *dest*, up to but not including the null terminator of *source*. If *source* contains less than *maxchar* characters, *dest* is padded with null characters. If *source* contains greater than or equal to *maxchar* characters, as many characters as possible are copied to *dest*. Be aware that the *dest* argument might not be terminated by a null character after a call to `strncpy`.

### Return Value

|   |                              |
|---|------------------------------|
| x | The address of <i>dest</i> . |
|---|------------------------------|



## Format

## Arguments

### Description

## Return Value

REF-501

## strpbrk

---

## strpbrk

Searches a string for the occurrence of one of a specified set of characters.

### Format

```
#include <string.h>

char *strpbrk (const char *str, const char *charset);
```

### Function Variants

The `strpbrk` function has variants named `_strpbrk32` and `_strpbrk64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **str**

A pointer to a character string. If this character string is a null string, 0 is returned.

#### **charset**

A pointer to a character string containing the set of characters for which the function will search.

### Description

The `strpbrk` function scans the characters in the string, stops when it encounters a character found in *charset*, and returns the address of the first character in the string that appears in the character set.

### Return Values

|      |                                                                      |
|------|----------------------------------------------------------------------|
| x    | The address of the first character in the string that is in the set. |
| NULL | Indicates that no character is in the set.                           |

---

## strptime

Converts a character string into date and time values that are stored in a `tm` structure. Conversion is controlled by a format string.

### Format

```
#include <time.h>
```

```
char *strptime (const char *buf, const char *format, struct tm *timeptr);
```

### Function Variants

The `strptime` function has variants named `_strptime32` and `_strptime64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**buf**

A pointer to the character string to convert.

**format**

A pointer to the string that defines how the input string is converted.

**timeptr**

A pointer to the local time structure. The `tm` structure is defined in the `<time.h>` header file.

### Description

The `strptime` function converts the string pointed to by *buf* into values that are stored in the structure pointed to by *timeptr*. The string pointed to by *format* defines how the conversion is performed.

The `strptime` function modifies only those fields in the `tm` structure that have corresponding conversion specifications in the format. In particular, `strptime` never sets the `tm_isdst` member of the `tm` structure.

The format string consists of zero or more directives. A directive is composed of one of the following:

- One or more white-space characters (as defined by the `isspace` function). This directive causes the function to read input up to the first character that is not a white-space character.
- Any character other than the percent character (`%`) or a white-space character. This directive causes the function to read the next character. The character read must be the same as the character that comprises the directive. If the character is different, the function fails.
- A conversion specification. A conversion specification defines how characters in the input string are interpreted as values that are then stored in the `tm` structure. A conversion specification consists of a percent (`%`) character followed by a conversion specifier. Table REF-9 lists the valid conversion specifications.

## strptime

The `strptime` function uses fields in the `LC_TIME` category of the program's current locale to provide a value.

---

### Note

---

To be compliant with X/Open CAE Specification System Interfaces and Headers Issue 5 (commonly known as XPG5), the `strptime` function processes the `"%y"` directive differently than in previous versions of the Compaq C RTL.

With Compaq C Version 6.4 and higher, for a two-digit year within the century if no century is specified, `"%y"` directive values ranging from:

- 69 to 99 refer to years in the twentieth century (1969 to 1999 inclusive)
- 00 to 68 refer to years in the twenty-first century (2000 to 2068 inclusive)

In previous (XPG4-compliant) versions of the Compaq C RTL, `strptime` interpreted a two-digit year with no century specified as a year within the twentieth century.

The XPG5-compliant `strptime` is now the default version in the Compaq C RTL.

To obtain the old, XPG4-compliant `strptime` function behavior, specify one of the following:

- Define the `DECC$XPG4_STRPTIME` logical name as follows:  

```
$ DEFINE DECC$XPG4_STRPTIME ENABLE
```

Or
- Call the XPG4 `strptime` directly as the function `decc$strptime_xpg4`.

To return to using the XPG5 `strptime` version, `DEASSIGN` the `DECC$XPG4_STRPTIME` logical name:

```
$ DEASSIGN DECC$XPG4_STRPTIME
```

---

**Table REF–9    `strptime` Conversion Specifications**

| Specification    | Replaced by                                                        |
|------------------|--------------------------------------------------------------------|
| <code>%a</code>  | The weekday name. This is either the abbreviated or the full name. |
| <code>%A</code>  | Same as <code>%a</code> .                                          |
| <code>%b</code>  | The month name. This is either the abbreviated or the full name.   |
| <code>%B</code>  | Same as <code>%b</code> .                                          |
| <code>%c</code>  | The date and time using the locale's date format.                  |
| <code>%Ec</code> | The locale's alternative date and time representation.             |

(continued on next page)

**Table REF–9 (Cont.) strptime Conversion Specifications**

| Specification | Replaced by                                                                                                                          |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------|
| %C            | The century number (the year divided by 100 and truncated to an integer) as a decimal number (00 – 99). Leading zeros are permitted. |
| %EC           | The name of the base year (period) in the locale's alternative representation.                                                       |
| %d            | The day of the month as a decimal number (01 – 31). Leading zeros are permitted.                                                     |
| %Od           | The day of the month using the locale's alternative numeric symbols.                                                                 |
| %D            | Same as %m/%d/%y.                                                                                                                    |
| %e            | Same as %d.                                                                                                                          |
| %Oe           | The date of the month using the locale's alternative numeric symbols.                                                                |
| %h            | Same as %b.                                                                                                                          |
| %H            | The hour (24-hour clock) as a decimal number (00 – 23). Leading zeros are permitted.                                                 |
| %OH           | The hour (24-hour clock) using the locale's alternative numeric symbols.                                                             |
| %I            | The hour (12-hour clock) as a decimal number (01 – 12). Leading zeros are permitted.                                                 |
| %OI           | The hour (12-hour clock) using the locale's alternative numeric symbols.                                                             |
| %j            | The day of the year as a decimal number (001 – 366).                                                                                 |
| %m            | The month as a decimal number (01 – 12). Leading zeros are permitted.                                                                |
| %Om           | The month using the locale's alternative numeric symbols.                                                                            |
| %M            | The minute as a decimal number (00 – 59). Leading zeros are permitted.                                                               |
| %OM           | The minutes using the locale's alternative numeric symbols.                                                                          |
| %n            | Any white-space character.                                                                                                           |
| %p            | The locale's equivalent of the AM/PM designations associated with a 12-hour clock.                                                   |
| %r            | The time in AM/PM notation (%I:%M:%S %p).                                                                                            |
| %R            | The time in 24-hour notation (%H:%M).                                                                                                |
| %S            | The second as a decimal number (00 – 61). Leading zeros are permitted.                                                               |
| %OS           | The seconds using the locale's alternative numeric symbols.                                                                          |
| %t            | Any white-space character.                                                                                                           |
| %T            | The time (%H:%M:%S).                                                                                                                 |

(continued on next page)

Table REF–9 (Cont.) strptime Conversion Specifications

| Specification | Replaced by                                                                                                                           |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------|
| %U            | The week number of the year (the first Sunday as the first day of week 1) as a decimal number (00 – 53). Leading zeros are permitted. |
| %OU           | The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.                     |
| %w            | The weekday as a decimal number (0 [Sunday] – 6). Leading zeros are permitted.                                                        |
| %Ow           | The weekday as a number (Sunday=0) using the locale's alternative numeric symbols.                                                    |
| %W            | The week number of the year (the first Monday as the first day of week 1) as a decimal number (00 – 53). Leading zeros are permitted. |
| %OW           | The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.                     |
| %x            | The locale's appropriate date representation.                                                                                         |
| %Ex           | The locale's alternative date representation.                                                                                         |
| %EX           | The locale's alternative time representation.                                                                                         |
| %X            | The locale's appropriate time representation.                                                                                         |
| %y            | The year without century as a decimal number (00 – 99).                                                                               |
| %Ey           | The offset from the base year (%EC) in the locale's alternative representation.                                                       |
| %Oy           | The year without the century using the locale's alternative numeric symbols.                                                          |
| %Y            | The year with century as a decimal number.                                                                                            |
| %EY           | The locale's full alternative year representation.                                                                                    |
| %%            | Literal % character.                                                                                                                  |

## Return Values

|      |                                                                                   |
|------|-----------------------------------------------------------------------------------|
| x    | A pointer to the character following the last character parsed.                   |
| NULL | Indicates that an error occurred. The contents of the tm structure are undefined. |

## Example

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <locale.h>
#include <errno.h>

#define NUM_OF_DATES 7
#define BUF_SIZE 256
```

```

/* This program takes a number of date and time strings and */
/* converts them into tm structs using strptime(). These tm */
/* structs are then passed to strftime() which will reverse the */
/* process. The resulting strings are then compared with the */
/* originals and if a difference is found then an error is */
/* displayed. */
main()
{
 int count,
 i;
 char buffer[BUF_SIZE];
 char *ret_val;
 struct tm time_struct;
 char dates[NUM_OF_DATES][BUF_SIZE] =
 {
 "Thursday 01 January 1970 00:08:20",
 "Tuesday 29 February 1972 08:26:40",
 "Tuesday 31 December 1991 23:59:59",
 "Wednesday 01 January 1992 00:00:00",
 "Sunday 03 May 1992 13:33:20",
 "Monday 04 May 1992 17:20:00",
 "Friday 15 May 1992 03:20:00"};

 for (i = 0; i < NUM_OF_DATES; i++) {
 /* Convert to a tm structure */
 ret_val = strptime(dates[i], "%A %d %B %Y %T", &time_struct);

 /* Check the return value */
 if (ret_val == (char *) NULL) {
 perror("strptime");
 exit(EXIT_FAILURE);
 }

 /* Convert the time structure back to a formatted string */
 count = strftime(buffer, BUF_SIZE, "%A %d %B %Y %T",
 &time_struct);

 /* Check the return value */
 if (count == 0) {
 perror("strftime");
 exit(EXIT_FAILURE);
 }

 /* Check the result */
 if (strcmp(buffer, dates[i]) != 0) {
 printf("Error: Converted string differs from the original\n");
 }
 else {
 printf("Successfully converted <%s>\n", dates[i]);
 }
 }
}

```

Running the example program produces the following result:

```

Successfully converted <Thursday 01 January 1970 00:08:20>
Successfully converted <Tuesday 29 February 1972 08:26:40>
Successfully converted <Tuesday 31 December 1991 23:59:59>
Successfully converted <Wednesday 01 January 1992 00:00:00>
Successfully converted <Sunday 03 May 1992 13:33:20>
Successfully converted <Monday 04 May 1992 17:20:00>
Successfully converted <Friday 15 May 1992 03:20:00>

```

## strrchr

---

## strrchr

Returns the address of the last occurrence of a given character in a null-terminated string. The terminating null character is considered to be part of the string.

### Format

```
#include <string.h>
```

```
char *strrchr (const char *str, int character);
```

### Function Variants

The `strrchr` function has variants named `_strrchr32` and `_strrchr64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**str**

A pointer to a null-terminated character string.

**character**

An object of type `int`.

### Description

See also `strchr` in this section.

### Return Values

|      |                                                                |
|------|----------------------------------------------------------------|
| x    | The address of the last occurrence of the specified character. |
| NULL | Indicates that the character does not occur in the string.     |



---

## strsep

Separates strings.

### Format

```
#include <string.h>

char *strsep (char **stringp, char *delim);
```

### Function Variants

The `strsep` function has variants named `_strsep32` and `_strsep64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **stringp**

A pointer to a pointer to a character string.

#### **delim**

A pointer to a string containing characters to be used as delimiters.

### Description

The `strsep` function locates in *stringp*, the first occurrence of any character in *delim* (or the terminating `'\0'` character) and replaces it with a `'\0'`. The location of the next character after the delimiter character (or `NULL`, if the end of the string is reached) is stored in the *stringp* argument. The original value of the *stringp* argument is returned.

You can detect an "empty" field; one caused by two adjacent delimiter characters, by comparing the location referenced by the pointer returned in the *stringp* argument to `'\0'`.

The *stringp* argument is initially `NULL`, `strsep` returns `NULL`.

### Return Values

|                   |                                                          |
|-------------------|----------------------------------------------------------|
| <code>x</code>    | The address of the string pointed to by <i>stringp</i> . |
| <code>NULL</code> | Indicates that <i>stringp</i> is <code>NULL</code> .     |

### Example

The following example uses `strsep` to parse a string, containing token delimited by white space, into an argument vector:

```
char **ap, **argv[10], *inputstring;

for (ap = argv; (*ap = strsep(&inputstring, " \t")) != NULL;)
 if (**ap != '\0')
 ++ap;
```

## strspn

---

## strspn

Returns the length of the prefix of a string that consists entirely of characters from a set of characters.

### Format

```
#include <string.h>

size_t strspn (const char *str, const char *charset);
```

### Arguments

**str**

A pointer to a character string. If this string is a null string, 0 is returned.

**charset**

A pointer to a character string containing the characters for which the function will search.

### Description

The `strspn` function scans the characters in the string, stops when it encounters a character not found in *charset*, and returns the length of the string's initial segment formed by characters found in *charset*.

### Return Value

|   |                            |
|---|----------------------------|
| x | The length of the segment. |
|---|----------------------------|

## strstr

Locates the first occurrence in the string pointed to by *s1* of the sequence of characters in the string pointed to by *s2*.

### Format

```
#include <string.h>

char *strstr (const char *s1, const char *s2);
```

### Function Variants

The `strstr` function has variants named `_strstr32` and `_strstr64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**s1, s2**  
Pointers to character strings.

### Return Values

|         |                                          |
|---------|------------------------------------------|
| Pointer | A pointer to the located string.         |
| NULL    | Indicates that the string was not found. |

### Example

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

main()
{
 static char lookin[]="that this is a test was at the end";

 putchar('\n');
 printf("String: %s\n", &lookin[0]);
 putchar('\n');
 printf("Addr: %s\n", &lookin[0]);
 printf("this: %s\n", strstr(&lookin[0] , "this"));
 printf("that: %s\n", strstr(&lookin[0] , "that"));
 printf("NULL: %s\n", strstr(&lookin[0] , ""));
 printf("was: %s\n", strstr(&lookin[0] , "was"));
 printf("at: %s\n", strstr(&lookin[0] , "at"));
 printf("the end: %s\n", strstr(&lookin[0] , "the end"));
 putchar('\n');

 exit(0);
}
```

This example produces the following results:

## strstr

```
$ RUN STRSTR EXAMPLE
String: that this is a test was at the end
Addr: that this is a test was at the end
this: this is a test was at the end
that: that this is a test was at the end
NULL: that this is a test was at the end
was: was at the end
at: at this is a test was at the end
the end: the end
$
```

---

## strtod

Converts a given string to a double-precision number.

### Format

```
#include <stdlib.h>

double strtod (const char *nptr, char **endptr);
```

### Function Variants

The `strtod` function has variants named `_strtod32` and `_strtod64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **nptr**

A pointer to the character string to be converted to a double-precision number.

#### **endptr**

The address of an object where the function can store the address of the first unrecognized character that terminates the scan. If *endptr* is a NULL pointer, the address of the first unrecognized character is not retained.

### Description

The `strtod` function recognizes an optional sequence of white-space characters (as defined by `isspace`), then an optional plus or minus sign, then a sequence of digits optionally containing a radix character, then an optional letter (e or E) followed by an optionally signed integer. The first unrecognized character ends the conversion.

The string is interpreted by the same rules used to interpret floating constants.

The radix character is defined the program's current locale (category `LC_NUMERIC`).

This function returns the converted value. For `strtod`, overflows are accounted for in the following manner:

- If the correct value causes an overflow, `HUGE_VAL` (with a plus or minus sign according to the sign of the value) is returned and `errno` is set to `ERANGE`.
- If the correct value causes an underflow, 0 is returned and `errno` is set to `ERANGE`.

If the string starts with an unrecognized character, then the conversion is not performed, *endptr* is set to *nptr*, a 0 value is returned, and `errno` is set to `EINVAL`.)

## strtod

### Return Values

|                |                                                                                                                                                                                                                                                                                                                              |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x              | The converted string.                                                                                                                                                                                                                                                                                                        |
| 0              | Indicates the conversion could not be performed.<br>errno is set to one of the following: <ul style="list-style-type: none"><li>• EINVAL - No conversion could be performed.</li><li>• ERANGE - The value would cause an underflow.</li><li>• ENOMEM - Not enough memory available for internal conversion buffer.</li></ul> |
| $\pm$ HUGE_VAL | Overflow occurred; errno is set to ERANGE.                                                                                                                                                                                                                                                                                   |

---

## strtok

Locates text tokens in a given string. The text tokens are delimited by one or more characters from a separator string that you specify. This function keeps track of its position in the string between calls and, as successive calls are made, the function works through the string, identifying the text token following the one identified by the previous call.

### Format

```
#include <string.h>

char *strtok (char *s1, const char *s2);
```

### Function Variants

The `strtok` function has variants named `_strtok32` and `_strtok64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **s1**

On the first call, a pointer to a string containing 0 or more text tokens. On all subsequent calls for that string, a NULL pointer.

#### **s2**

A pointer to a separator string consisting of one or more characters. The separator string may differ from call to call.

### Description

A token in *s1* starts at the first character that is not a character in the separator string *s2* and ends either at the end of the string or at (but not including) a separator character.

The first call to the `strtok` function returns a pointer to the first character in the first token and writes a null character into *s1* immediately following the returned token. Each subsequent call (with the value of the first argument remaining NULL) returns a pointer to a subsequent token in the string originally pointed to by *s1*. When no tokens remain in the string, the `strtok` function returns a NULL pointer. (This can occur on the first call to `strtok` if the string is empty or contains only separator characters.)

Since `strtok` inserts null characters into *s1* to delimit tokens, *s1* cannot be a const object.

### Return Values

|      |                                                             |
|------|-------------------------------------------------------------|
| x    | A pointer to the first character of a token.                |
| NULL | Indicates that there are no tokens remaining in <i>s1</i> . |

## strtok

### Examples

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

 main()
 {
 static char str[] = "...ab..cd,,ef.hi";

 printf("%s\n", strtok(str, "."));
 printf("%s\n", strtok(NULL, ","));
 printf("%s\n", strtok(NULL, ",."));
 printf("%s\n", strtok(NULL, ",."));
 }
```

Running this example program produces the following results:

```
$ RUN STRTOK_EXAMPLE1
|ab|
|.cd|
|ef|
|hi|
$
```

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

 main()
 {
 char *ptr,
 string[30];

 /* The first character not in the string "-" is "A". The */
 /* token ends at "C". */
 strcpy(string, "ABC");
 ptr = strtok(string, "-");
 printf("%s\n", ptr);

 /* Returns NULL because no characters not in separator */
 /* string "-" were found (i.e. only separator characters */
 /* were found) */
 strcpy(string, "-");
 ptr = strtok(string, "-");
 if (ptr == NULL)
 printf("ptr is NULL\n");
 }
```

Running this example program produces the following results:

```
$ RUN STRTOK_EXAMPLE2
|abc|
ptr is NULL
$
```



---

## strtol

Converts strings of ASCII characters to the appropriate numeric values.

### Format

```
#include <stdlib.h>

long int strtol (const char *nptr, char **endptr, int base);
```

### Function Variants

The `strtol` function has variants named `_strtol32` and `_strtol64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **nptr**

A pointer to the character string to be converted to a long.

#### **endptr**

The address of an object where the function can store a pointer to the first unrecognized character encountered in the conversion process (that is, the character that follows the last character in the string being converted). If *endptr* is a NULL pointer, the address of the first unrecognized character is not retained.

#### **base**

The value, 2 through 36, to use as the base for the conversion.

### Description

The `strtol` function recognizes strings in various formats, depending on the value of the base. This function ignores any leading white-space characters (as defined by `isspace` in `<ctype.h>`) in the given string. It recognizes an optional plus or minus sign, then a sequence of digits or letters that may represent an integer constant according to the value of the base. The first unrecognized character ends the conversion.

Leading zeros after the optional sign are ignored, and `0x` or `0X` is ignored if the base is 16.

If *base* is 0, the sequence of characters is interpreted by the same rules used to interpret an integer constant: after the optional sign, a leading 0 indicates octal conversion, a leading `0x` or `0X` indicates hexadecimal conversion, and any other combination of leading characters indicates decimal conversion.

Truncation from long to int can take place after assignment or by an explicit cast (arithmetic exceptions notwithstanding). The function call `atoi(str)` is equivalent to `strtol(str, (char**)NULL, 10)`.

## strtol

### Return Values

|                      |                                                                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x                    | The converted value.                                                                                                                                                                                      |
| LONG_MAX or LONG_MIN | Indicates that the converted value would cause an overflow.                                                                                                                                               |
| 0                    | Indicates that the string starts with an unrecognized character or that the value for <i>base</i> is invalid. If the string starts with an unrecognized character, <i>*endptr</i> is set to <i>nptr</i> . |

---

**strtq, strtoll** *(Alpha only)*

Converts strings of ASCII characters to the appropriate numeric values. `strtoll` is a synonym for `strtq`.

**Format**

```
#include <stdlib.h>

__int64 strtq (const char *nptr, char **endptr, int base);
__int64 strtoll (const char *nptr, char **endptr, int base);
```

**Function Variants**

This function has variants named `_strtq32/_strtoll32` and `_strtq64/_strtoll64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

**Arguments****nptr**

A pointer to the character string to be converted to an `__int64`.

**endptr**

The address of an object where the function can store a pointer to the first unrecognized character encountered in the conversion process (that is, the character that follows the last character in the string being converted). If `endptr` is a NULL pointer, the address of the first unrecognized character is not retained.

**base**

The value, 2 through 36, to use as the base for the conversion.

**Description**

This function recognizes strings in various formats, depending on the value of the base. This function ignores any leading white-space characters (as defined by `isspace` in `<ctype.h>`) in the given string. It recognizes an optional plus or minus sign, then a sequence of digits or letters that may represent an integer constant according to the value of the base. The first unrecognized character ends the conversion.

Leading zeros after the optional sign are ignored, and `0x` or `0X` is ignored if the base is 16.

If `base` is 0, the sequence of characters is interpreted by the same rules used to interpret an integer constant: after the optional sign, a leading 0 indicates octal conversion, a leading `0x` or `0X` indicates hexadecimal conversion, and any other combination of leading characters indicates decimal conversion.

The function call `atq(str)` is equivalent to `strtq(str, (char**)NULL, 10)`.

**strtoq, strtoll** *(Alpha only)*

## Return Values

|                                                         |                                                                                                                                                                                                           |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>                                          | The converted value.                                                                                                                                                                                      |
| <code>__INT64_MAX</code> or<br><code>__INT64_MIN</code> | Indicates that the converted value would cause an overflow.                                                                                                                                               |
| <code>0</code>                                          | Indicates that the string starts with an unrecognized character or that the value for <i>base</i> is invalid. If the string starts with an unrecognized character, <i>*endptr</i> is set to <i>nptr</i> . |

---

## strtol

Converts the initial portion of the string pointed to by *nptr* to an unsigned long integer.

### Format

```
#include <stdlib.h>
```

```
unsigned long int strtol (const char *nptr, char **endptr, int base);
```

### Function Variants

The `strtol` function has variants named `_strtol32` and `_strtol64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**nptr**

A pointer to the character string to be converted to an unsigned long.

**endptr**

The address of an object where the function can store a pointer to a pointer to the first unrecognized character encountered in the conversion process (that is, the character that follows the last character in the string being converted). If *endptr* is a NULL pointer, the address of the first unrecognized character is not retained.

**base**

The value, 2 through 36, to use as the base for the conversion. Leading zeros after the optional sign are ignored, and 0x or 0X is ignored if the base is 16.

If the base is 0, the sequence of characters is interpreted by the same rules used to interpret an integer constant: after the optional sign, a leading 0 indicates octal conversion, a leading 0x or 0X indicates hexadecimal conversion, and any other combination of leading characters indicates decimal conversion.

### Return Values

x

The converted value.

0

Indicates that the string starts with an unrecognized character or that the value for *base* is invalid. If the string starts with an unrecognized character, *\*endptr* is set to *nptr*.

ULONG\_MAX

Indicates that the converted value would cause an overflow.

---

## strtouq, strtoull (Alpha only)

Converts the initial portion of the string pointed to by *nptr* to an unsigned `__int64` integer. `strtoull` is a synonym for `strtouq`.

### Format

```
#include <stdlib.h>

unsigned __int64 strtouq (const char *nptr, char **endptr, int base);
unsigned __int64 strtoull (const char *nptr, char **endptr, int base);
```

### Function Variants

This function has variants named `_strtouq32/_strtoull32` and `_strtouq64/_strtoull64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **nptr**

A pointer to the character string to be converted to an unsigned `__int64`.

#### **endptr**

The address of an object where the function can store a pointer to a pointer to the first unrecognized character encountered in the conversion process (that is, the character that follows the last character in the string being converted). If *endptr* is a NULL pointer, the address of the first unrecognized character is not retained.

#### **base**

The value, 2 through 36, to use as the base for the conversion. Leading zeros after the optional sign are ignored, and 0x or 0X is ignored if the base is 16.

If the base is 0, the sequence of characters is interpreted by the same rules used to interpret an integer constant: after the optional sign, a leading 0 indicates octal conversion, a leading 0x or 0X indicates hexadecimal conversion, and any other combination of leading characters indicates decimal conversion.

### Return Values

|              |                                                                                                                                                                                                           |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x            | The converted value.                                                                                                                                                                                      |
| 0            | Indicates that the string starts with an unrecognized character or that the value for <i>base</i> is invalid. If the string starts with an unrecognized character, <i>*endptr</i> is set to <i>nptr</i> . |
| __UINT64_MAX | Indicates that the converted value would cause an overflow.                                                                                                                                               |

---

## strxfrm

Changes a string such that the changed string can be passed to the `strcmp` function and produce the same result as passing the unchanged string to the `strcoll` function.

### Format

```
#include <string.h>

size_t strxfrm (char *s1, const char *s2, size_t maxchar);
```

### Arguments

**s1, s2**

Pointers to character strings.

**maxchar**

The maximum number of bytes (including the null terminator) to be stored in *s1*.

### Description

The `strxfrm` function transforms the string pointed to by *s2*, and stores the resulting string in the array pointed to by *s1*. No more than *maxchar* bytes, including the null terminator, are placed into the array pointed to by *s1*.

If the value of *maxchar* is less than the required size to store the transformed string (including the terminating null), the contents of the array pointed to by *s1* is indeterminate. In such a case, the function returns the size of the transformed string.

If *maxchar* is 0, then *s1* is allowed to be a NULL pointer, and the function returns the required size of the *s1* array before making the transformation.

The string comparison functions, `strcoll` and `strcmp`, can produce different results given the same two string to compare. The reason for this is that `strcmp` does a straightforward comparison of the code point values of the characters in the strings, whereas `strcoll` uses the locale information to do the comparison. Depending on the locale, the `strcoll` comparison can be a multipass operation, which is slower than `strcmp`.

The purpose of the `strxfrm` function is to transform strings in such a way that if you pass two transformed strings to the `strcmp` function, the result is the same as passing the two original strings to the `strcoll` function. The `strxfrm` function is useful in applications that need to do a large number of comparisons on the same strings using `strcoll`. In this case, it might be more efficient (depending on the locale) to transform the strings once using `strxfrm`, and then do comparisons using `strcmp`.

## strxfrm

### Return Value

x                      Length of the resulting string pointed to by *s1*, not including the terminating null character. No return value is reserved for error indication. However, the function can set *errno* to *EINVAL* – The string pointed to by *ws2* contains characters outside the domain of the collating sequence.

### Example

```
/* This program verifies that two transformed strings when */
/* passed through strxfrm and then compared, provide the same */
/* result as if passed through strcoll without any */
/* transformation. */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

#define BUFF_SIZE 256

main()
{
 char string1[BUFF_SIZE];
 char string2[BUFF_SIZE];
 int errno;
 int coll_result;
 int strcmp_result;
 size_t strxfrm_result1;
 size_t strxfrm_result2;

 /* setlocale to French locale */

 if (setlocale(LC_ALL, "fr_FR.ISO8859-1") == NULL) {
 perror("setlocale");
 exit(EXIT_FAILURE);
 }

 /* collate string 1 and string 2 and store the result */

 errno = 0;
 coll_result = strcoll("<a'>bcd", "abcz");
 if (errno) {
 perror("strcoll");
 exit(EXIT_FAILURE);
 }

 else {
 /* Transform the strings (using strxfrm) into string1 */
 /* and string2 */

 strxfrm_result1 = strxfrm(string1, "<a'>bcd", BUFF_SIZE);

 if (strxfrm_result1 == ((size_t) - 1)) {
 perror("strxfrm");
 exit(EXIT_FAILURE);
 }

 else if (strxfrm_result1 > BUFF_SIZE) {
 perror("\n**String is too long **\n");
 exit(EXIT_FAILURE);
 }
 }
}
```



```

else {
 strxfrm_result2 = strxfrm(string2, "abcz", BUFF_SIZE);
 if (strxfrm_result2 == ((size_t) - 1)) {
 perror("strxfrm");
 exit(EXIT_FAILURE);
 }
 else if (strxfrm_result2 > BUFF_SIZE) {
 perror("\n** String is too long **\n");
 exit(EXIT_FAILURE);
 }
 /* Compare the two transformed strings and verify */
 /* that the result is the same as the result from */
 /* strcoll on the original strings */
 else {
 strcmp_result = strcmp(string1, string2);
 if (strcmp_result == 0 && (coll_result == 0)) {
 printf("\nReturn value from strcoll() and "
 "return value from strcmp() are both zero.");
 printf("\nThe program was successful\n\n");
 }
 else if ((strcmp_result < 0) && (coll_result < 0)) {
 printf("\nReturn value from strcoll() and "
 "return value from strcmp() are less than zero.");
 printf("\nThe program successful\n\n");
 }
 else if ((strcmp_result > 0) && (coll_result > 0)) {
 printf("\nReturn value from strcoll() and return "
 "value from strcmp() are greater than zero.");
 printf("\nThe program was successful\n\n");
 }
 else {
 printf("*** Error **\n");
 printf("\nReturn values are not of the same type");
 }
 }
}
}
}

```

Running the example program produces the following result:

```

Return value from strcoll() and return value
 from strcmp() are less than zero.
The program was successful

```

## subwin

---

## subwin

Creates a new subwindow with *numlines* lines and *numcols* columns starting at the coordinates (*begin\_y*,*begin\_x*) on the terminal screen.

### Format

```
#include <curses.h>
```

```
WINDOW *subwin (WINDOW *win, int numlines, int numcols, int begin_y, int begin_x);
```

### Arguments

#### **win**

A pointer to the parent window.

#### **numlines**

The number of lines in the subwindow. If *numlines* is 0, then the function sets that dimension to `LINES - begin_y`. To get a subwindow of dimensions `LINES` by `COLS`, use the following format:

```
subwin (win, 0, 0, 0, 0)
```

#### **numcols**

The number of columns in the subwindow. If *numcols* is 0, then the function sets that dimension to `COLS - begin_x`. To get a subwindow of dimensions `LINES` by `COLS`, use the following format:

```
subwin (win, 0, 0, 0, 0)
```

#### **begin\_y**

A window coordinate at which the subwindow is to be created.

#### **begin\_x**

A window coordinate at which the subwindow is to be created.

### Description

When creating the subwindow, *begin\_y* and *begin\_x* are relative to the entire terminal screen. If either *numlines* or *numcols* is 0, then the subwin function sets that dimension to `(LINES - begin_y)` or `(COLS - begin_x)`, respectively.

The window pointed to by *win* must be large enough to contain the entire area of the subwindow. Any changes made to either window within the coordinates of the subwindow appear on both windows.

### Return Values

window pointer

A pointer to an instance of the structure window corresponding to the newly created subwindow.

ERR

Indicates an error.

---

## swab

Swaps bytes.

### Format

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

```
void swab (const void *src, void *dest, ssize_t nbytes);
```

### Arguments

**src**

A pointer to the location of the string to copy.

**dest**

A pointer to where you want the results copied.

**nbytes**

The number of bytes to copy. Make this argument an even value. When it is an odd value, the swab function uses  $nbytes - 1$  instead.

### Description

The swab function copies the number of bytes specified by *nbytes* from the location pointed to by *src* to the array pointed to by *dest*. The function then exchanges adjacent bytes. If a copy takes place between objects that overlap, the result is undefined.

## swprintf

---

## swprintf

Writes output to an array of wide characters under control of the wide-character format string.

### Format

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

```
int swprintf (wchar_t *s, size_t n, const wchar_t *format, ...);
```

### Arguments

**s**

A pointer to the resulting wide-character sequence.

**n**

The maximum number of wide characters that can be written to an array pointed to by *s*, including a terminating null wide character.

**format**

A pointer to a wide-character string containing the format specifications. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

**...**

Optional expressions whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, the output sources can be omitted. Otherwise, the function calls must have exactly as many output sources as there are conversion specifications, and the conversion specifications must match the types of the output sources.

Conversion specifications are matched to output sources in left-to-right order. Excess output pointers, if any, are ignored.

### Description

The `swprintf` function is equivalent to the `fwprintf` function, except that the first argument specifies an array of wide characters instead of a stream.

No more than *n* wide characters are written, including a terminating null wide character, which is always added (unless *n* is 0).

See also `fwprintf` in this section.

### Return Values

**x**

The number of wide characters written, not counting the terminating null wide character.

Negative value

Indicates an error. Either *n* or more wide characters were requested to be written, or a conversion error occurred, in which case `errno` is set to `EILSEQ`.

---

## swscanf

Reads input from a wide-character string under control of the wide-character format string.

### Format

```
#include <wchar.h>

int swscanf (const wchar_t *s, const wchar_t *format, ...);
```

### Arguments

**s**

A pointer to a wide-character string from which the input is to be obtained.

**format**

A pointer to a wide-character string containing the format specifications. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

**...**

Optional expressions whose results correspond to conversion specifications given in the format specification.

If no conversion specifications are given, you can omit the input pointers. Otherwise, the function calls must have exactly as many input pointers as there are conversion specifications, and the conversion specifications must match the types of the input pointers.

Conversion specifications are matched to input sources in left-to-right order. Excess input pointers, if any, are ignored.

### Description

The `swscanf` function is equivalent to the `fwscanf` function, except that the first argument specifies a wide-character string rather than a stream. Reaching the end of the wide-character string is the same as encountering EOF for the `fwscanf` function.

See also `fwscanf` in this section.

### Return Values

|                |                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>x</code> | The number of input items assigned, sometimes fewer than provided for, or even 0 in the event of an early matching failure. |
| EOF            | Indicates an error. An input failure occurred before any conversion.                                                        |

## sysconf

---

## sysconf

Gets configurable system variables.

### Format

```
#include <unistd.h>

long int sysconf (int name);
```

### Argument

**name**  
Specifies the system variable to be queried.

### Description

The `sysconf` function provides a method for determining the current value of a configurable system limit or whether optional features are supported.

You supply a symbolic constant in the *name* argument, and `sysconf` returns a value for the corresponding system variable:

- The symbolic constants defined in the `<unistd.h>` header file.
- The system variables are defined in the `<limits.h>` and `<unistd.h>` header files.

Table REF–10 lists the system variables returned by the `sysconf` function, and the symbolic constants that you can supply as the *name* value.

**Table REF–10 SYSCONF Argument and Return Values**

| System Variable Returned | Symbolic Constant for <i>name</i> | Meaning                                                                                                                                                      |
|--------------------------|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ISO POSIX-1</b>       |                                   |                                                                                                                                                              |
| ARG_MAX                  | _SC_ARG_MAX                       | The maximum length, in bytes, of the arguments for one of the <code>exec</code> functions, including environment data.                                       |
| CHILD_MAX                | _SC_CHILD_MAX                     | The maximum number of simultaneous processes for each real user ID.                                                                                          |
| CLK_TCK                  | _SC_CLK_TCK                       | The number of clock ticks per second. The value of <code>CLK_TCK</code> can be variable. Do not assume that <code>CLK_TCK</code> is a compile time constant. |
| NGROUPS_MAX              | _SC_NGROUPS_MAX                   | The maximum number of simultaneous supplementary group IDs for each process.                                                                                 |
| OPEN_MAX                 | _SC_OPEN_MAX                      | The maximum number of files that one process can have open at one time.                                                                                      |

(continued on next page)

Table REF–10 (Cont.) SYSCONF Argument and Return Values

| System Variable Returned | Symbolic Constant for<br><i>name</i> | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ISO POSIX-1</b>       |                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| STREAM_MAX               | _SC_STREAM_MAX                       | The number of streams that one process can have open at one time.                                                                                                                                                                                                                                                                                                                                                                                    |
| TZNAME_MAX               | _SC_TZNAME_MAX                       | The maximum number of bytes supported for the name of a time zone (not the length of the TZ environmental variable).                                                                                                                                                                                                                                                                                                                                 |
| _POSIX_JOB_CONTROL       | _SC_JOB_CONTROL                      | This variable has a value of 1 if the system supports job control; otherwise, –1 is returned.                                                                                                                                                                                                                                                                                                                                                        |
| _POSIX_SAVED_IDS         | _SC_SAVED_IDS                        | This variable has a value of 1 if each process has a saved set user ID and a saved set group ID; otherwise, –1 is returned.                                                                                                                                                                                                                                                                                                                          |
| _POSIX_VERSION           | _SC_VERSION                          | The date of approval of the most current version of the POSIX-1 standard that the system supports. The date is a 6-digit number, with the first 4 digits signifying the year and the last 2 digits the month. If _POSIX_VERSION is not defined, –1 is returned.<br><br>Different versions of the POSIX-1 standard are periodically approved by the IEEE Standards Board, and the date of approval is used to distinguish between different versions. |
| <b>ISO POSIX-2</b>       |                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| BC_BASE_MAX              | _SC_BC_BASE_MAX                      | The maximum value allowed for the obase variable with the bc command.                                                                                                                                                                                                                                                                                                                                                                                |
| BC_DIM_MAX               | _SC_BC_DIM_MAX                       | The maximum number of elements permitted in an array by the bc command.                                                                                                                                                                                                                                                                                                                                                                              |
| BC_SCALE_MAX             | _SC_BC_SCALE_MAX                     | The maximum value allowed for the scale variable with the bc command.                                                                                                                                                                                                                                                                                                                                                                                |
| BC_STRING_MAX            | _SC_BC_STRING_MAX                    | The maximum length of string constants accepted by the bc command.                                                                                                                                                                                                                                                                                                                                                                                   |
| COLL_WEIGHTS_MAX         | _SC_COLL_WEIGHTS_MAX                 | The maximum number of weights that can be assigned to an entry in the LC_COLLATE locale-dependent information in a locale definition file.                                                                                                                                                                                                                                                                                                           |

(continued on next page)

## sysconf

Table REF–10 (Cont.) SYSCONF Argument and Return Values

| System Variable Returned | Symbolic Constant for<br><i>name</i> | Meaning                                                                                                                                                                                                             |
|--------------------------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ISO POSIX-2</b>       |                                      |                                                                                                                                                                                                                     |
| EXPR_NEST_MAX            | _SC_EXPR_NEST_MAX                    | The maximum number of expressions that you can nest within parentheses by the <code>expr</code> command.                                                                                                            |
| LINE_MAX                 | _SC_LINE_MAX                         | The maximum length, in bytes, of a command input line (either standard input or another file) when the utility is described as processing text files. The length includes room for the trailing new-line character. |
| RE_DUP_MAX               | _SC_RE_DUP_MAX                       | The maximum number of repeated occurrences of a regular expression permitted when using the interval notation arguments, such as the <i>m</i> and <i>n</i> arguments with the <code>ed</code> command.              |
| _POSIX2_CHAR_TERM        | _SC_2_CHAR_TERM                      | This variable has a value of 1 if the system supports at least one terminal type; otherwise, <code>-1</code> is returned.                                                                                           |
| _POSIX2_C_BIND           | _SC_2_C_BIND                         | This variable has a value of 1 if the system supports the C language binding option; otherwise, <code>-1</code> is returned.                                                                                        |
| _POSIX2_C_DEV            | _SC_2_C_DEV                          | This variable has a value of 1 if the system supports the optional C Language Development Utilities from the ISO POSIX-2 standard; otherwise, <code>-1</code> is returned.                                          |
| _POSIX2_C_VERSION        | _SC_2_C_VERSION                      | Integer value indicating the version of the ISO POSIX-2 standard (C language binding). It changes with each new version of the ISO POSIX-2 standard.                                                                |
| _POSIX2_VERSION          | _SC_2_VERSION                        | Integer value indicating the version of the ISO POSIX-2 standard (Commands). It changes with each new version of the ISO POSIX-2 standard.                                                                          |
| _POSIX2_FORT_DEV         | _SC_2_FORT_DEV                       | The variable has a value of 1 if the system supports the Fortran Development Utilities Option from the ISO POSIX-2 standard; otherwise, <code>-1</code> is returned.                                                |
| _POSIX2_FORT_RUN         | _SC_2_FORT_RUN                       | The variable has a value of 1 if the system supports the Fortran Runtime Utilities Option from the ISO POSIX-2 standard; otherwise, <code>-1</code> is returned.                                                    |

(continued on next page)



Table REF–10 (Cont.) SYSCONF Argument and Return Values

| System Variable Returned                       | Symbolic Constant for<br><i>name</i>          | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ISO POSIX-2</b>                             |                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>_POSIX2_LOCALEDEF</code>                 | <code>_SC_2_LOCALEDEF</code>                  | The variable has a value of 1 if the system supports the creation of new locales with the <code>localedef</code> command; otherwise, <code>-1</code> is returned.                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>_POSIX2_SW_DEV</code>                    | <code>_SC_2_SW_DEV</code>                     | The variable has a value of 1 if the system supports the Software Development Utilities Option from the ISO POSIX-2 standard; otherwise, <code>-1</code> is returned.                                                                                                                                                                                                                                                                                                                                                                                          |
| <code>_POSIX2_UPE</code>                       | <code>_SC_2_UPE</code>                        | The variable has a value of 1 if the system supports the User Portability Utilities Option; otherwise, <code>-1</code> is returned.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>POSIX 1003.1c-1995</b>                      |                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>_POSIX_THREADS</code>                    | <code>_SC_THREADS</code>                      | This variable has a value of 1 if the system supports POSIX threads; otherwise, <code>-1</code> is returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <code>_POSIX_THREAD_ATTR_STACKSIZE</code>      | <code>_SC_THREAD_ATTR_STACKSIZE</code>        | This variable has a value of 1 if the system supports the POSIX threads stack size attribute; otherwise, <code>-1</code> is returned.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <code>_POSIX_THREAD_PRIORITY_SCHEDULING</code> | <code>_SC_THREAD_PRIORITY_SCHEDULING</code>   | The 1003.1c implementation supports the realtime scheduling functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>_POSIX_THREAD_SAFE_FUNCTIONS</code>      | <code>_SC_THREAD_SAFE_FUNCTIONS</code>        | TRUE if the implementation supports the thread-safe ANSI C functions in POSIX 1003.1c.                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>PTHREAD_DESTRUCTOR_ITERATIONS</code>     | <code>_SC_THREAD_DESTRUCTOR_ITERATIONS</code> | When a thread terminates, <code>DECthreads</code> iterates through all non-NULL thread-specific data values in the thread, and calls a registered destructor routine (if any) for each. It is possible for a destructor routine to create new values for one or more thread-specific data keys. In that case, <code>DECthreads</code> goes through the entire process again.<br><code>_SC_THREAD_DESTRUCTOR_ITERATIONS</code> is the maximum number of times the implementation loops before it terminates the thread even if there are still non-NULL values. |

(continued on next page)

## sysconf

Table REF–10 (Cont.) SYSCONF Argument and Return Values

| System Variable Returned  | Symbolic Constant for<br><i>name</i> | Meaning                                                                                                                                    |
|---------------------------|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>POSIX 1003.1c-1995</b> |                                      |                                                                                                                                            |
| PTHREAD_KEYS_MAX          | _SC_THREAD_KEYS_MAX                  | The maximum number of thread-specific data keys that an application can create.                                                            |
| PTHREAD_STACK_MIN         | _SC_THREAD_STACK_MIN                 | The minimum allowed size of a stack for a new thread. Any lower value specified for the "stacksize" thread attribute is rounded up.        |
| UINT_MAX                  | _SC_THREAD_THREADS_MAX               | The maximum number of threads an application is allowed to create. Since DECthreads does not enforce any fixed limit, this value is $-1$ . |
| <b>X/Open</b>             |                                      |                                                                                                                                            |
| _XOPEN_VERSION            | _SC_XOPEN_VERSION                    | An integer indicating the most current version of the X/Open standard that the system supports.                                            |
| PASS_MAX                  | _SC_PASS_MAX                         | Maximum number of significant bytes in a password (not including terminating null).                                                        |
| XOPEN_CRYPT               | _SC_XOPEN_CRYPT                      | This variable has a value of 1 if the system supports the X/Open Encryption Feature Group; otherwise, $-1$ is returned.                    |
| XOPEN_ENH_I18N            | _SC_XOPEN_ENH_I18N                   | This variable has a value of 1 if the system supports the X/Open enhanced Internationalization Feature Group; otherwise, $-1$ is returned. |
| XOPEN_SHM                 | _SC_XOPEN_SHM                        | This variable has a value of 1 if the system supports the X/Open Shared Memory Feature Group; otherwise, $-1$ is returned.                 |
| <b>X/Open Extended</b>    |                                      |                                                                                                                                            |
| ATEXIT_MAX                | _SC_ATEXIT_MAX                       | The maximum number of functions that you can register with <code>atexit</code> per process.                                                |
| PAGESIZE                  | _SC_PAGESIZE                         | Size in bytes of a page.                                                                                                                   |
| PAGE_SIZE                 | _SC_PAGE_SIZE                        | Same as PAGESIZE. If either PAGESIZE or PAGE_SIZE is defined, the other is defined with the same value.                                    |

(continued on next page)

Table REF–10 (Cont.) SYSCONF Argument and Return Values

| System Variable Returned | Symbolic Constant for<br><i>name</i> | Meaning                                                                                                                                                                                                     |
|--------------------------|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>X/Open Extended</b>   |                                      |                                                                                                                                                                                                             |
| IOV_MAX                  | _SC_IOV_MAX                          | Maximum number of iovec structures that one process has available for use with readv or writev.                                                                                                             |
| XOPEN_UNIX               | _SC_XOPEN_UNIX                       | This variable has a value of 1 if the system supports the X/Open CAE Specification, August 1994, System Interfaces and Headers, Issue 4, Version 2, (ISBN: 1-85912-037-7, C435); otherwise, –1 is returned. |

Return Values

|    |                                                                                                                                                                                               |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | The current variable value on the system. The value does not change during the lifetime of the calling process.                                                                               |
| –1 | Indicates an error.<br>If the value of the <i>name</i> argument is invalid, errno is set to indicate the error.<br>If the value of the <i>name</i> argument is undefined, errno is unchanged. |

## system

---

## system

Passes a given string to the host environment to be executed by a command processor. This function is nonreentrant.

### Format

```
#include <stdlib.h>

int system (const char *string);
```

### Argument

#### **string**

A pointer to the string to be executed. If *string* is NULL, a nonzero value is returned. The string is a DCL command, not the name of an image. To execute an image, use one of the exec routines.

### Description

The system function spawns a subprocess and executes the command specified by *string* in that subprocess. The system function waits for the subprocess to complete before returning the subprocess status as the return value of the function.

The subprocess is spawned within the system call by a call to vfork. Because of this, a call to system should not be made after a call to vfork and before the corresponding call to an exec function.

For OpenVMS Version 7.0 and higher systems, if you include <stdlib.h> and compile with the \_POSIX\_EXIT feature-test macro set, then the system function returns the status as if it called waitpid to wait for the child. Therefore, use the WIFEXITED and WEXITSTATUS macros to retrieve the exit status in the range of 0 to 255.

You set the \_POSIX\_EXIT feature-test macro by using /DEFINE=\_POSIX\_EXIT or #define \_POSIX\_EXIT at the top of your file, before any file inclusions.

### Return Value

|               |                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| nonzero value | If <i>string</i> is NULL, a value of 1 is returned, indicating that the system function is supported. If <i>string</i> is not NULL, the value is the subprocess OpenVMS return status. |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Example

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h> /* write, close */
#include <fcntl.h> /* Creat */

main()
{
 int status,
 fd;

 /* Creat a file we are sure is there */
 /*
```

```
fd = creat("system.test", 0);
write(fd, "this is an example of using system", 34);
close(fd);

if (system(NULL)) {
 status = system("DIR/NOHEAD/NOTRAIL/SIZE SYSTEM.TEST");
 printf("system status = %d\n", status);
}
else
 printf("system() not supported.\n");
}
```

Running this example program produces the following result:

```
DISK3$:[JONES.CRTL.2059.SRC]SYSTEM.TEST;1
1
system status = 1
```

## tan

---

## tan

Returns a double value that is the tangent of its radian argument.

### Format

```
#include <math.h>

double tan (double x);
float tanf (float x); (Alpha only)
long double tanl (long double x); (Alpha only)
double tand (double x); (Alpha only)
float tandf (float x); (Alpha only)
long double tandl (long double x); (Alpha only)
```

### Argument

**x**  
A radian expressed as a real number.

### Description

The tan functions compute the tangent of  $x$ , measured in radians.

The tand functions compute the tangent of  $x$ , measured in degrees.

### Return Values

|                                         |                                                                   |
|-----------------------------------------|-------------------------------------------------------------------|
| <b>x</b>                                | The tangent of the argument.                                      |
| <b>HUGE_VAL</b>                         | $x$ is a singular point ( $\dots -3\pi/2, -\pi/2, \pi/2 \dots$ ). |
| <b>NaN</b>                              | $x$ is NaN; errno is set to EDOM.                                 |
| <b>0</b>                                | $x$ is $\pm\text{Infinity}$ ; errno is set to EDOM.               |
| <b><math>\pm\text{HUGE\_VAL}</math></b> | Overflow occurred; errno is set to ERANGE.                        |
| <b>0</b>                                | Underflow occurred; errno is set to ERANGE.                       |

---

## tanh

Returns the hyperbolic tangent of its argument.

### Format

```
#include <math.h>

double tanh (double x);
float tanhf (float x); (Alpha only)
long double tanhl (long double x); (Alpha only)
```

### Argument

**x**  
A real number.

### Description

The tanh functions return the hyperbolic tangent their argument, calculated as  $(e^{**x} - e^{**(-x)})/(e^{**x} + e^{**(-x)})$ .

### Return Values

|          |                                                    |
|----------|----------------------------------------------------|
| n        | The hyperbolic tangent of the argument.            |
| HUGE_VAL | The argument is too large; errno is set to ERANGE. |
| NaN      | $x$ is NaN; errno is set to EDOM.                  |
| 0        | Underflow occurred; errno is set to ERANGE.        |

## telldir

---

## telldir

Returns the current location associated with a specified directory stream.  
Performs operations on directories.

### Format

```
#include <dirent.h>

long int telldir (DIR *dir_pointer);
```

### Argument

**dir\_pointer**

A pointer to the DIR structure of an open directory.

### Description

The telldir function returns the current location associated with the specified directory stream.

### Return Values

|    |                                                                  |
|----|------------------------------------------------------------------|
| x  | The current location.                                            |
| −1 | Indicates an error and is further specified in the global errno. |



---

## tempnam

Constructs the name for a temporary file.

### Format

```
#include <stdio.h>

char *tempnam (const char *directory, const char *prefix);
```

### Arguments

**directory**

A pointer to the pathname of the directory where you want to create a file.

**prefix**

A pointer to an initial character sequence of the file name. The *prefix* argument can be null, or it can point to a string of up to five characters used as the first characters of the temporary file name.

### Description

The `tempnam` function generates file names for temporary files. It allows you to control the choice of a directory.

If the *directory* argument is null or points to a string that is not a pathname for an appropriate directory, the pathname defined as `P_tmpdir` in the `<stdio.h>` header file is used.

You can bypass the selection of a pathname by providing the `TMPDIR` environment variable in the user environment. The value of the `TMPDIR` variable is a pathname for the desired temporary file directory.

Use the *prefix* argument to specify a prefix of up to 5 characters for the temporary file name.

The `tempnam` function returns a pointer to the generated pathname, suitable for use in a subsequent call to the `free` function. See also `free` in this section.

---

**Note**

---

In contrast to `tmpnam`, `tempnam` does not have to generate a different file name on each call. `tempnam` generates a new file name only if the file with the specified name exists. If you need a unique file name on each call, use `tmpnam` instead of `tempnam`.

---

### Return Values

x

A pointer to the generated pathname, suitable for use in a subsequent call to the `free` function.

NULL

An error occurred; `errno` is set to indicate the error.

## time

---

## time

Returns the time (expressed as Universal Coordinated Time) elapsed since 00:00:00, January 1, 1970, in seconds.

### Format

```
#include <time.h>

time_t time (time_t *time_location);
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `time` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Argument

#### **time\_location**

Either `NULL` or a pointer to the place where the returned time is also stored. The `time_t` type is defined in the `<time.h>` header file as follows:

```
typedef unsigned long int time_t;
```

### Return Values

|                           |                                                                                                                                                                           |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>            | The time elapsed past the Epoch.                                                                                                                                          |
| <code>(time_t)(-1)</code> | Indicates an error. If the value of <code>SYS\$TIMEZONE_DIFFERENTIAL</code> logical is wrong, the function will fail with <code>errno</code> set to <code>EINVAL</code> . |

---

## times

Passes back the accumulated times of the current process and its terminated child processes.

### Format

```
#include <times.h>

clock_t times (struct tms *buffer); (OpenVMS 7.0 and higher)

void times (tbuffer_t *buffer); (pre OpenVMS 7.0)
```

### Argument

**buffer**

A pointer to the terminal buffer.

### Description

For both process and children times, the structure breaks down the time by user and system time. Since the OpenVMS system does not differentiate between system and user time, all system times are returned as 0. Accumulated CPU times are returned in 10-millisecond units.

Only the accumulated times for child processes running a C main program or a program that calls VAXC\$CRTL\_INIT or DECC\$CRTL\_INIT are included.

On OpenVMS Version 7.0 and higher systems, the times function returns the elapsed real time in clock ticks since an arbitrary reference time in the past (for example, system startup time). This reference time does not change from one times function call to another. The return value can overflow the possible range of type clock\_t values. When times fails, it returns a value of -1. The Compaq C RTL uses system-boot time as its reference time.

### Return Values

|               |                                                              |
|---------------|--------------------------------------------------------------|
| x             | The elapsed real time in clock ticks since system-boot time. |
| (clock_t)(-1) | Indicates an error.                                          |

## tmpfile

---

## tmpfile

Creates a temporary file that is opened for update.

### Format

```
#include <stdio.h>
FILE *tmpfile (void);
```

### Description

The file exists only for the duration of the process, or until the file is closed and is preserved across calls to `vfork`.

### Return Values

|                   |                                                                                          |
|-------------------|------------------------------------------------------------------------------------------|
| <code>x</code>    | The address of a file pointer (defined in the <code>&lt;stdio.h&gt;</code> header file). |
| <code>NULL</code> | Indicates an error.                                                                      |

---

## tmpnam

Generates file names that can be safely used for a temporary file.

### Format

```
#include <stdio.h>

char *tmpnam (char *name);
```

### Function Variants

The tmpnam function has variants named `_tmpnam32` and `_tmpnam64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Argument

#### **name**

A character string containing a name to use in place of file-name arguments in functions or macros. Successive calls to tmpnam with a null argument cause the function to overwrite the current name.

### Return Value

|   |                                                                                                                                                                                                                                                                                                                                                              |
|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x | If the <i>name</i> argument is the null pointer value NULL, tmpnam returns the address of an internal storage area. If <i>name</i> is not NULL, then it is considered the address of an area of length <code>L_tmpnam</code> (defined in the <code>&lt;stdio.h&gt;</code> header file). In this case, tmpnam returns the <i>name</i> argument as the result. |
|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**toascii****toascii**

Converts its argument, an 8-bit ASCII character, to a 7-bit ASCII character.

## Format

```
#include <ctype.h>

int toascii (char character);
```

## Argument

**character**  
An object of type char.

## Return Value

|   |                          |
|---|--------------------------|
| x | A 7-bit ASCII character. |
|---|--------------------------|

---

## tolower

Converts a character to lowercase.

### Format

```
#include <ctype.h>

int tolower (int character);
```

### Argument

#### **character**

An object of type `int` representable as an unsigned `char` or the value of `EOF`. For any other value, the behavior is undefined.

### Description

If the argument represents an uppercase letter, and there is a corresponding lowercase letter, as defined by character type information in the program locale category `LC_TYPE`, the corresponding lowercase letter is returned.

If the argument is not an uppercase character, it is returned unchanged.

### Return Value

|   |                                                                                                                      |
|---|----------------------------------------------------------------------------------------------------------------------|
| x | The lowercase letter corresponding to the argument. Or, the unchanged argument, if it is not an uppercase character. |
|---|----------------------------------------------------------------------------------------------------------------------|

## **\_tolower**

---

## **\_tolower**

Converts an uppercase character to lowercase.

### **Format**

```
#include <ctype.h>
int _tolower (int character);
```

### **Argument**

**character**  
This argument must be an uppercase letter.

### **Description**

The `_tolower` macro is equivalent to the `tolower` function except that its argument *must* be an uppercase letter (not lowercase, not EOF).

The `_tolower` macro should not be used with arguments that contain side-effect operations. For instance, the following example will not return the expected result:

```
d = _tolower (c++);
```

### **Return Value**

|   |                                                     |
|---|-----------------------------------------------------|
| x | The lowercase letter corresponding to the argument. |
|---|-----------------------------------------------------|



---

## touchwin

Places the most recently edited version of the specified window on the terminal screen.

### Format

```
#include <curses.h>
int touchwin (WINDOW *win);
```

### Argument

**win**  
A pointer to the window.

### Description

The touchwin function is normally used only to refresh overlapping windows.

### Return Values

|     |                     |
|-----|---------------------|
| OK  | Indicates success.  |
| ERR | Indicates an error. |

## toupper

---

## toupper

Converts a character to uppercase.

### Format

```
#include <ctype.h>

int toupper (int character);
```

### Argument

#### **character**

An object of type `int` representable as an unsigned `char` or the value of `EOF`. For any other value, the behavior is undefined.

### Description

If the argument represents a lowercase letter, and there is a corresponding uppercase letter, as defined by character type information in the program locale category `LC_TYPE`, the corresponding uppercase letter is returned.

If the argument is not a lowercase character, it is returned unchanged.

### Return Value

|   |                                                                                                                               |
|---|-------------------------------------------------------------------------------------------------------------------------------|
| x | The uppercase letter corresponding to the argument. Or, the unchanged argument, if the argument is not a lowercase character. |
|---|-------------------------------------------------------------------------------------------------------------------------------|

---

## **\_toupper**

Converts a lowercase character to uppercase.

### **Format**

```
#include <ctype.h>
int _toupper (int character);
```

### **Argument**

#### **character**

This argument must be an uppercase letter.

### **Description**

The `_toupper` macro is equivalent to the `toupper` function except that its argument *must* be a lowercase letter (not uppercase, not EOF).

The `_toupper` macro should not be used with arguments that contain side-effect operations. For instance, the following example will not return the expected result:

```
d = _toupper (c++);
```

### **Return Value**

x

The uppercase letter corresponding to the argument.

## towctrans

---

## towctrans

Maps one wide character to another according to a specified mapping descriptor.

### Format

```
#include <wctype.h>

wint_t towctrans (wint_t wc, wctrans_t desc);
```

### Arguments

**wc**  
The wide character that you want to map.

**desc**  
Description of the mapping obtained through a call to the `wctrans` function.

### Description

The `towctrans` function maps the wide character specified in *wc*, using the mapping described by *desc*.

The current setting of the `LC_CTYPE` category must be the same as during the call to the `wctrans` function that returned the value of *desc*.

### Return Values

|          |                                                                                                                                                                     |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>x</b> | The mapped value of the <i>wc</i> wide character, if this character exists in the mapping described by <i>desc</i> . Otherwise, the value of <i>wc</i> is returned. |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

## tolower

Converts the argument, a wide-character code, to lowercase. If the argument is not an uppercase character, it is returned unchanged.

### Format

```
#include <wctype.h> (ISO C)
```

```
#include <wchar.h> (XPG4)
```

```
int towlower (wint_t wc);
```

### Argument

**wc**

An object of type `wint_t` representable as a valid wide character in the current locale, or the value of `WEOF`. For any other value, the behavior is undefined.

### Description

If the argument is an uppercase wide character, the corresponding lowercase wide character (as defined in the `LC_CTYPE` category of the locale) is returned, if it exists. If it does not exist, the function returns the input argument unchanged.

## towupper

---

## towupper

Converts the argument, a wide character, to uppercase. If the argument is not a lowercase character, it is returned unchanged.

### Format

```
#include <wctype.h> (ISO C)
#include <wchar.h> (XPG4)

int towupper (wint_t wc);
```

### Argument

**wc**

An object of type `wint_t` representable as a valid wide character in the current locale, or the value of `WEOF`. For any other value, the behavior is undefined.

### Description

If the argument is a lowercase wide character, the corresponding uppercase wide character (as defined in the `LC_CTYPE` category of the locale) is returned, if it exists. If it does not exist, the function returns the input argument unchanged.

**trunc** (*Alpha only*)

Truncates the argument to an integral value.

## Format

```
#include <math.h>

double trunc (double x);

float truncf (float x);

long double truncL (long double x);
```

## Argument

**x**  
A floating-point number.

## Return Value

|   |                                                |
|---|------------------------------------------------|
| n | The truncated, integral value of the argument. |
|---|------------------------------------------------|

## truncate

---

## truncate

Changes file length to a specified length in bytes.

### Format

```
#include <unistd.h>

int truncate (const char *path, off_t length);
```

### Arguments

#### **path**

The name of a file that is to be truncated. This argument must point to a pathname that names a regular file for which the calling process has write permission.

#### **length**

The new length of the file in bytes. The `off_t` type of *length* is either a 64-bit integer or a 32-bit integer. The 64-bit interface allows for file sizes greater than 2 gigabytes, and can be selected at compile time by defining the `_LARGEFILE` feature-test macro as follows:

```
CC/DEFINE=_LARGEFILE
```

### Description

The `truncate` function changes the length of a file to the size in bytes specified by the *length* argument.

If the new length is less than the previous length, the function removes all data beyond *length* bytes from the specified file. All file data between the new End-of-File and the previous End-of-File is discarded.

For stream files, if the new length is greater than the previous length, new file data between the previous End-of-File and the new End-of-File is added, consisting of all zeros. (For record files, it is not possible to extend the file in this manner.)

### Return Values

|    |                                                                     |
|----|---------------------------------------------------------------------|
| 0  | Indicates success.                                                  |
| −1 | An error occurred; <code>errno</code> is set to indicate the error. |



---

## **ttyname**

Returns a pointer to the null-terminated name of the terminal device associated with file descriptor 0, the default input device (stdin).

### **Format**

```
#include <unixio.h>

char *ttyname (void);
```

### **Description**

The `ttyname` function is provided only for UNIX compatibility and has limited use in the OpenVMS environment.

### **Return Values**

|   |                                                |
|---|------------------------------------------------|
| x | A pointer to a null-terminated string.         |
| 0 | Indicates that SYS\$INPUT is not a TTY device. |

---

**tzset**

Sets and accesses time-zone conversion.

**Format**

```
#include <time.h>

void tzset (void);

extern char *tzname[];

extern long int timezone;

extern int daylight;
```

**Description**

The `tzset` function initializes time conversion information used by the `ctime`, `localtime`, `mktime`, `strftime`, and `wcsftime` functions.

The `tzset` function sets the following external variables:

- `tzname` is set as follows, where "std" is a 3-byte name for the standard time zone, and "dst" is a 3-byte name for the Daylight Savings Time zone:

```
tzname[0] = "std"
tzname[1] = "dst"
```
- `daylight` is set to 0 if Daylight Savings Time should never be applied to the time zone. Otherwise, `daylight` is set to 1.
- `timezone` is set to the difference between UTC and local standard time.

The environment variable `TZ` specifies how `tzset` initializes time conversion information:

- If `TZ` is absent from the environment, the implementation-dependent time-zone information is used, as follows:

The best available approximation to local wall-clock time is used, as defined by the system logical `SYS$LOCALTIME`, which points to a `tzfile` format file that describes default time-zone rules. This system logical is set during the installation of OpenVMS Version 7.0 or higher to define a time-zone file based off the root directory `SYS$COMMON:[SYS$ZONEINFO.SYSTEM]`.<sup>1</sup>
- If `TZ` appears in the environment but its value is a null string, Coordinated Universal Time (UTC) is used (without leap-second correction).

---

<sup>1</sup> The *Compaq C* RTL uses a public-domain, time-zone handling package that puts time-zone conversion rules in easily accessible and modifiable files. These files reside in the directory `SYS$COMMON:[SYS$ZONEINFO.SYSTEM.SOURCES]`. The time-zone compiler `zic` converts these files to a special format described by the `<tzfile.h>` header file. The converted files are created with a root directory of `SYS$COMMON:[SYS$ZONEINFO.SYSTEM]`, which is pointed to by the `SYS$TZDIR` system logical. This format is readable by the C library functions that handle time-zone information. For example, in the eastern United States, `SYS$LOCALTIME` is defined to be `SYS$COMMON:[SYS$ZONEINFO.SYSTEM.US]EASTERN`.

- If TZ appears in the environment and its value is not a null string, the value has one of three formats, as described in Table REF–11.

Table REF–11 Time-Zone Initialization Rules

| TZ Format                                                            | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| :                                                                    | UTC is used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| : <i>pathname</i>                                                    | The characters following the colon specify the pathname of a tzfile format file from which to read the time-conversion information. A pathname beginning with a slash (/) represents an absolute pathname; otherwise, the pathname is relative to the system time-conversion information directory specified by SYS\$TZDIR, which by default is SYS\$COMMON:[SYS\$ZONEINFO.SYSTEM].                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <i>stdoffset</i> [ <i>dst</i> [ <i>offset</i> ]<br>[, <i>rule</i> ]] | <p>The value is first used as the pathname of a file (as described for the <i>:pathname</i> format) from which to read the time-conversion information.</p> <p>If that file cannot be read, the value is then interpreted as a direct specification of the time-conversion information, as follows:</p> <p><i>std</i> and <i>dst</i>—Three or more characters that are the designation for the time zone:</p> <ul style="list-style-type: none"><li>• <i>std</i>—Standard time zone. Required.</li><li>• <i>dst</i>—Daylight Savings Time zone. Optional. If <i>dst</i> is omitted, Daylight Savings Time does not apply.</li></ul> <p>Uppercase and lowercase letters are explicitly allowed. Any characters are allowed, except the following:</p> <ul style="list-style-type: none"><li>• digits</li><li>• leading colon ( : )</li><li>• comma ( , )</li><li>• minus ( - )</li><li>• plus ( + )</li><li>• ASCII null character</li></ul> <p><i>offset</i>—The value added to the local time to arrive at UTC. The offset has the following format:</p> <p><i>hh</i>[ :<i>mm</i>[ :<i>ss</i> ]]</p> <p>In this format:</p> <ul style="list-style-type: none"><li>• <i>hh</i> (hours) is a one-or two-digit value of 0–24.</li><li>• <i>mm</i> (minutes) is a value of 0–59. (optional)</li><li>• <i>ss</i> (seconds) is a value of 0–59. (optional)</li></ul> |

(continued on next page)

Table REF–11 (Cont.) Time-Zone Initialization Rules

| TZ Format | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           | <p>The offset following <i>std</i> is required. If no offset follows <i>dst</i>, summer time is assumed, one hour ahead of standard time. You can use one or more digits; the value is always interpreted as a decimal number.</p> <p>If the time zone is preceded by a minus sign (–), the time zone is East of Greenwich; otherwise, it is West, which can also be indicated by a preceding plus sign (+).</p> <p><i>rule</i>—Indicates when to change to and return from summer time. The rule has the form:</p> <p><i>start[/time], end[/time]</i></p> <p>Where:</p> <ul style="list-style-type: none"> <li>• <i>start</i> is the date when the change from standard time to summer time occurs.</li> <li>• <i>end</i> is the date for returning from summer time to standard time.</li> </ul> <p>If <i>start</i> and <i>end</i> are omitted, the default is the US Daylight Savings Time start and end dates. The format for <i>start</i> and <i>end</i> must be one of the following:</p> <ul style="list-style-type: none"> <li>• <i>Jn</i>—The Julian day <i>n</i> (<math>1 &lt; n &lt; 365</math>). Leap days are not counted. That is, in all years, including leap years, February 28 is day 59 and March 1 is day 60. You cannot explicitly refer to February 29.</li> <li>• <i>n</i>—The zero based Julian day (<math>0 &lt; n &lt; 365</math>). Leap days are counted, making it possible to refer to February 29.</li> <li>• <i>Mm.n.d</i>—The <i>n</i>th <i>d</i> day of month <i>m</i> where: <ul style="list-style-type: none"> <li><math>0 &lt; n &lt; 5</math></li> <li><math>0 &lt; d &lt; 6</math></li> <li><math>1 &lt; m &lt; 12</math></li> </ul> <p>When <i>n</i> is 5, it refers to the last <i>d</i> day of month <i>m</i>. Sunday is day 0.</p> </li> </ul> |

(continued on next page)

Table REF–11 (Cont.) Time-Zone Initialization Rules

| TZ Format | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           | <p><i>time</i>—The time when, in current time, the change to or return from summer time occurs. The <i>time</i> argument has the same format as <i>offset</i>, except that you cannot use a leading minus (–) or plus (+) sign. If <i>time</i> is not specified, the default is 02:00:00.</p> <p>If no rule is present in the TZ specification, the rules used are those specified by the tzfile format file defined by the system logical SYS\$POSIXRULES in the system time-conversion information directory, with the standard and summer time offsets from UTC replaced by those specified by the offset values in TZ.</p> <p>If TZ does not specify a tzfile format file and cannot be interpreted as a direct specification, UTC is used.</p> |

**Note**

The UTC-based time functions, introduced in OpenVMS Version 7.0, had degraded performance compared with the non-UTC-based time functions. OpenVMS Version 7.1 added a cache for time-zone files to improve performance. The size of the cache is determined by the logical name DECC\$TZ\_CACHE\_SIZE. To accommodate most countries changing the time twice per year, the default cache size is large enough to hold two time-zone files.

See also ctime, localtime, mktime, strftime, and wcsftime in this section.

**Sample TZ Specification**

EST5EDT4,M4.1.0,M10.5.0

This sample TZ specification describes the rule defined in 1987 for the Eastern time zone in the US:

- EST (Eastern Standard Time) is the designation for standard time, which is 5 hours behind UTC.
- EDT (Eastern Daylight Time) is the designation for summer time, which is 4 hours behind UTC. EDT starts on the first Sunday in April and ends on the last Sunday in October.

Because *time* was not specified in either case, the changes occur at the default time, which is 2:00 A.M. The start and end dates did not need to be specified, because they are the defaults.

## ualarm

---

## ualarm

Sets or changes the timeout of interval timers.

### Format

```
#include <unistd.h>

useconds_t ualarm (useconds_t mseconds, useconds_t interval);
```

### Arguments

#### **mseconds**

Specifies a number of real time microseconds.

#### **interval**

Specifies the interval for repeating the timer.

### Description

The `ualarm` function causes the `SIGALRM` signal to be generated for the calling process after the number of real-time microseconds specified by *useconds* has elapsed. When the *interval* argument is nonzero, repeated timeout notification occurs with a period in microseconds specified by *interval*. If the notification signal `SIGALRM` is not caught or is ignored, the calling process is terminated.

If you call a combination of `ualarm` and `setitimer` functions, and the AST status is disabled, the return value is invalid.

If you call a combination of `ualarm` and `setitimer` functions, and the AST status is enabled, the return value is valid.

This is because you cannot invoke an AST handler to clear the previous value of the timer when ASTs are disabled or invoked from a handler that was invoked at AST level.

---

#### **Note**

Interactions between `ualarm` and either `alarm`, or `sleep` are unspecified.

---

See also `setitimer` in this section.

### Return Values

|                 |                                                                                                            |
|-----------------|------------------------------------------------------------------------------------------------------------|
| <code>n</code>  | The number of microseconds remaining from the previous <code>ualarm</code> or <code>setitimer</code> call. |
| <code>0</code>  | No timeouts are pending or <code>ualarm</code> not previously called.                                      |
| <code>-1</code> | Indicates an error.                                                                                        |

---

## umask

Creates a file protection mask that is used when a new file is created, and returns the previous mask value.

### Format

```
#include <stat.h>

mode_t umask (mode_t mode_complement);
```

### Argument

#### **mode\_complement**

Shows which bits to turn off when a new file is created. See the description of `chmod` to determine what the bits represent.

### Description

Initially, the file protection mask is set from the current process's default file protection. This is done when the C main program starts up or when `DECC$CRTL_INIT` (or `VAXC$CRTL_INIT`) is called. You can change this for all files created by your program by calling `umask` or you can use `chmod` to change the file protection on individual files. The file protection of a file created by `open` or `creat` is the bitwise AND of the `open` and `creat` mode argument with the complement of the value passed to `umask` on the previous call.

---

#### **Note**

---

The way to create files with OpenVMS RMS default protections using the UNIX system-call functions `umask`, `mkdir`, `creat`, and `open` is to call `mkdir`, `creat`, and `open` with a file-protection mode argument of 0777 in a program that never specifically calls `umask`. These default protections include correctly establishing protections based on ACLs, previous versions of files, and so on.

In programs that do `vfork/exec` calls, the new process image inherits whether `umask` has ever been called or not from the calling process image. The `umask` setting and whether the `umask` function has ever been called are both inherited attributes.

---

### Return Value

x

The old mask value.

## uname

---

## uname

Gets system identification information.

### Format

```
#include <utsname.h>

int uname (struct utsname *name);
```

### Arguments

**name**  
The current system identifier.

### Description

The uname function stores null-terminated strings of information identifying the current system into the structure referenced by the *name* argument.

The utsname structure is defined in the <utsname.h> header file and contains the following members:

|          |                                             |
|----------|---------------------------------------------|
| sysname  | Name of the operating system implementation |
| nodename | Network name of this machine                |
| release  | Release level of the operating system       |
| version  | Version level of the operating system       |
| machine  | Machine hardware platform                   |

### Return Values

|    |                                                                 |
|----|-----------------------------------------------------------------|
| 0  | Indicates success.                                              |
| −1 | Indicates an error; errno or vaxc\$errno is set as appropriate. |



---

## ungetc

Pushes a character back into the input stream and leaves the stream positioned before the character.

### Format

```
#include <stdio.h>

int ungetc (int character, FILE *file_ptr);
```

### Arguments

**character**

A value of type `int`.

**file\_ptr**

A file pointer.

### Description

When using the `ungetc` function, the character is pushed back onto the file indicated by *file\_ptr*.

One push-back is guaranteed, even if there has been no previous activity on the file. The `fseek` function erases all memory of pushed-back characters. The pushed-back character is not written to the underlying file. If the character to be pushed back is EOF, the operation fails, the input stream is left unchanged, and EOF is returned.

See also `fseek` and `getc` in this section.

### Return Values

|     |                                              |
|-----|----------------------------------------------|
| x   | The push-back character.                     |
| EOF | Indicates it cannot push the character back. |

## ungetwc

---

## ungetwc

Pushes a wide character back into the input stream.

### Format

```
#include <wchar.h>

wint_t ungetwc (wint_t wc, FILE *file_ptr);
```

### Arguments

**wc**  
A value of type `wint_t`.

**file\_ptr**  
A file pointer.

### Description

When using the `ungetwc` function, the wide character is pushed back onto the file indicated by *file\_ptr*.

One push-back is guaranteed, even if there has been no previous activity on the file. If a file positioning function (such as `fseek`) is called before the pushed back character is read, the bytes representing the pushed back character are lost.

If the character to be pushed back is `WEOF`, the operation fails, the input stream is left unchanged, and `WEOF` is returned.

See also `getwc` in this section.

### Return Values

|                   |                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>    | The push-back character.                                                                                                                                                                                                                                                                                                                                                           |
| <code>WEOF</code> | Indicates that the function cannot push the character back. <code>errno</code> is set to one of the following: <ul style="list-style-type: none"><li>• <code>EBADF</code> – The file descriptor is not valid.</li><li>• <code>EALREADY</code> – Operation is already in progress on the same file.</li><li>• <code>EILSEQ</code> – Invalid wide-character code detected.</li></ul> |

---

**unordered** *(Alpha only)*

Returns the value 1 (TRUE) if either or both of the arguments is a NaN.  
Otherwise, it returns the value 0 (FALSE).

**Format**

```
#include <math.h>

double unordered (double x, double y);
float unorderedf (float x, float y);
long double unorderedl (long double x, long double y);
```

**Arguments**

**x**  
A real number.

**y**  
A real number.

**Return Values**

|   |                                           |
|---|-------------------------------------------|
| 1 | Either or both of the arguments is a NaN. |
| 0 | Neither argument is a NaN.                |

---

## utime

Sets file access and modification times.

### Format

```
#include <types.h>

int utime (const char *path, const struct utimbuf *times);
```

### Arguments

#### **path**

A pointer to a file.

#### **times**

A null pointer or a pointer to a utimbuf structure.

### Description

The `utime` function sets the access and modification times of the file named by the *path* argument.

If *times* is a null pointer, the access and modification times of the file are set to the current time. To use `utime` in this way, the effective user ID of the process must match the owner of the file, or the process must have write permission to the file or have appropriate privileges.

If *times* is not a null pointer, it is interpreted as a pointer to a `utimbuf` structure, and the access and modification times are set to the values in the specified structure. Only a process with an effective user ID equal to the user ID of the file or a process with appropriate privileges can use `utime` this way.

The `utimbuf` structure is defined by the `<utime.h>` header. The times in the `utimbuf` structure are measured in seconds since the Epoch.

Upon successful completion, `utime` marks the time of the last file status change, *st\_ctime*, to be updated. See the `<stat.h>` header file.

---

#### **Note** *(Alpha only)*

---

On OpenVMS Alpha systems, the `stat`, `fstat`, `utime`, and `utimes` functions have been enhanced to take advantage of the new file-system support for POSIX-compliant file timestamps.

This support is available only on ODS-5 devices on OpenVMS Alpha systems beginning with a version of OpenVMS Alpha after Version 7.3.

Before this change, `stat` and `fstat` set the values of the *st\_ctime*, *st\_mtime*, and *st\_atime* fields based on the following file attributes:

- st\_ctime* - `ATR$C_CREDATE` (file creation time)
- st\_mtime* - `ATR$C_REVDATE` (file revision time)
- st\_atime* - was always set to *st\_mtime* because no support for file access time was available

Also, for the file-modification time, `utime` and `utimes` were modifying the `ATR$C_REVDATE` file attribute, and ignoring the file-access-time argument.

After the change, for a file on an ODS-5 device, the `stat` and `fstat` functions set the values of the `st_ctime`, `st_mtime`, and `st_atime` fields based on the following new file attributes:

```
st_ctime - ATR$C_ATTDATE (last attribute modification time)
st_mtime - ATR$C_MODDATE (last data modification time)
st_atime - ATR$C_ACCDATE (last access time)
```

If `ATR$C_ACCDATE` is zero, as on an ODS-2 device, the `stat` and `fstat` functions set `st_atime` to `st_mtime`.

For the file-modification time, the `utime` and `utimes` functions modify both the `ATR$C_REVDATE` and `ATR$C_MODDATE` file attributes. For the file-access time, these functions modify the `ATR$C_ACCDATE` file attribute. Setting the `ATR$C_MODDATE` and `ATR$C_ACCDATE` file attributes on an ODS-2 device has no effect.

For compatibility, the old behavior of `stat`, `fstat`, `utime`, and `utimes` remains the default, regardless of the kind of device.

The new behavior must be explicitly enabled by defining the `DECC$EFS_FILE_TIMESTAMPS` logical name to "ENABLE" before invoking the application. Setting this logical does not affect the behavior of `stat`, `fstat`, `utime`, and `utimes` for files on an ODS-2 device.

---

## Return Values

|   |                       |
|---|-----------------------|
| 0 | Successful execution. |
|---|-----------------------|

– 1

Indicates an error. The function sets `errno` to one of the following values:

The `utime` function *will* fail if:

- `EACCES` – Search permission is denied by a component of the *path* prefix; or the *times* argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied.
- `ELOOP` – Too many symbolic links were encountered in resolving *path*.
- `ENAMETOOLONG` – The length of the *path* argument exceeds `{PATH_MAX}`, a pathname component is longer than `{NAME_MAX}`, or a pathname resolution of a symbolic link produced an intermediate result whose length exceeds `{PATH_MAX}`.
- `ENOENT` – *path* does not name an existing file, or *path* is an empty string.
- `ENOTDIR` – A component of the *path* prefix is not a directory.
- `EPERM` – *times* is not a null pointer and the calling process' effective user ID has write-access to the file but does not match the owner of the file, and the calling process does not have the appropriate privileges.
- `EROFS` – The file system containing the file is read-only.

---

## utimes

Sets file access and modification times.

### Format

```
#include <time.h>

int utimes (const char *path, const struct timeval times[2]);
```

### Arguments

#### **path**

A pointer to a file.

#### **times**

an array of `timeval` structures. The first array member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the `timeval` structure are measured in seconds and microseconds since the Epoch, although rounding toward the nearest second may occur.

### Description

The `utimes` function sets the access and modification times of the file pointed to by the *path* argument to the value of the *times* argument. The `utimes` function allows time specifications accurate to the microsecond.

If the *times* argument is a null pointer, the access and modification times of the file are set to the current time. The effective user ID of the process must be the same as the owner of the file, or must have write access to the file or appropriate privileges to use this call in this manner.

Upon completion, `utimes` marks the time of the last file status change, `st_ctime`, for update.

---

#### **Note** (*Alpha only*)

---

On OpenVMS Alpha systems, the `stat`, `fstat`, `utime`, and `utimes` functions have been enhanced to take advantage of the new file-system support for POSIX-compliant file timestamps.

This support is available only on ODS-5 devices on OpenVMS Alpha systems beginning with a version of OpenVMS Alpha after Version 7.3.

Before this change, the `stat` and `fstat` functions were setting the values of the `st_ctime`, `st_mtime`, and `st_atime` fields based on the following file attributes:

- `st_ctime` - `ATR$C_CREDATE` (file creation time)
- `st_mtime` - `ATR$C_REVDATE` (file revision time)
- `st_atime` - was always set to `st_mtime` because no support for file access time was available

Also, for the file-modification time, `utime` and `utimes` were modifying the `ATR$C_REVDATE` file attribute, and ignoring the file-access-time argument.

## utimes

After the change, for a file on an ODS-5 device, the `stat` and `fstat` functions set the values of the `st_ctime`, `st_mtime`, and `st_atime` fields based on the following new file attributes:

- `st_ctime` - `ATR$C_ATTDATE` (last attribute modification time)
- `st_mtime` - `ATR$C_MODDATE` (last data modification time)
- `st_atime` - `ATR$C_ACCDATE` (last access time)

If `ATR$C_ACCDATE` is zero, as on an ODS-2 device, the `stat` and `fstat` functions set `st_atime` to `st_mtime`.

For the file-modification time, the `utime` and `utimes` functions modify both the `ATR$C_REVDATE` and `ATR$C_MODDATE` file attributes. For the file-access time, these functions modify the `ATR$C_ACCDATE` file attribute. Setting the `ATR$C_MODDATE` and `ATR$C_ACCDATE` file attributes on an ODS-2 device has no effect.

For compatibility, the old behavior of `stat`, `fstat`, `utime`, and `utimes` remains the default, regardless of the kind of device.

The new behavior must be explicitly enabled by defining the `DECC$EFS_FILE_TIMESTAMPS` logical name to "ENABLE" before invoking the application. Setting this logical does not affect the behavior of `stat`, `fstat`, `utime`, and `utimes` for files on an ODS-2 device.

---

## Return Values

|   |                       |
|---|-----------------------|
| 0 | Successful execution. |
|---|-----------------------|



-1

Indicates an error. The file times do not change and the function sets `errno` to one of the following values:

The `utimes` function *will* fail if:

- `EACCES` – Search permission is denied by a component of the *path* prefix; or the *times* argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied.
- `ELOOP` – Too many symbolic links were encountered in resolving *path*.
- `ENAMETOOLONG` – The length of the *path* argument exceeds `{PATH_MAX}`, a pathname component is longer than `{NAME_MAX}`, or a pathname resolution of a symbolic link produced an intermediate result whose length exceeds `{PATH_MAX}`.
- `ENOENT` – A component of *path* does not name an existing file, or *path* is an empty string.
- `ENOTDIR` – A component of the *path* prefix is not a directory.
- `EPERM` – The *times* argument is not a null pointer and the calling process' effective user ID has write-access to the file but does not match the owner of the file and the calling process does not have the appropriate privileges.
- `EROFS` – The file system containing the file is read-only.

## unsetenv

---

## unsetenv

Deletes all instances of the environment variable *name* from the environment list.

### Format

```
#include <stdlib.h>

void unsetenv (const char *name);
```

### Argument

**name**

The environment variable to delete from the environment list.

### Description

The `unsetenv` function deletes all instances of the variable *name* pointed to by the *name* argument from the environment list.

---

## usleep

Suspends execution for an interval.

### Format

```
#include <unistd.h>

int usleep (unsigned int mseconds);
```

### Argument

#### **mseconds**

The number of microseconds to suspend execution for.

### Description

The `usleep` function suspends the current process from execution for the number of microseconds specified by the *mseconds* argument. This argument must be less than 1,000,000. However, if its value is 0, then the call has no effect.

There is one real-time interval timer for each process. The `usleep` function does not interfere with a previous setting of this timer. If the process set this timer before calling `usleep` and if the time specified by *mseconds* equals or exceeds the interval timer's prior setting, then the process is awakened shortly before the timer was set to expire.

### Return Values

|    |                                                                                 |
|----|---------------------------------------------------------------------------------|
| 0  | Indicates success.                                                              |
| −1 | Indicates an error occurred; <code>errno</code> is set to <code>EINVAL</code> . |

---

## VAXC\$CRTL\_INIT

Allows you to call the *Compaq C* RTL from other languages or to use the *Compaq C* RTL when your main function is not in C. It initializes the run-time environment and establishes both an exit and condition handler. VAXC\$CRTL\_INIT is a synonym for DECC\$CRTL\_INIT. Either name invokes the same routine.

### Format

```
#include <signal.h>

void VAXC$CRTL_INIT();
```

### Description

The following example shows a Pascal program that calls the *Compaq C* RTL using the VAXC\$CRTL\_INIT function:

On OpenVMS VAX systems:

```
$ PASCAL EXAMPLE
$ LINK EXAMPLE,SYS$LIBRARY:DECCRTL/LIB
$ TY EXAMPLE.PAS
PROGRAM TESTC(input, output);
PROCEDURE VAXC$CRTL_INIT; extern;
BEGIN
 VAXC$CRTL_INIT;
END
$
```

On OpenVMS Alpha systems:

```
$ PASCAL EXAMPLE
$ LINK EXAMPLE,SYS$LIBRARY:VAXCRTL/LIB
$ TY EXAMPLE.PAS
PROGRAM TESTC(input, output);
PROCEDURE VAXC$CRTL_INIT; extern;
BEGIN
 VAXC$CRTL_INIT;
END
$
```

A shareable image need only call this function if it contains a *Compaq C* function for signal handling, environment variables, I/O, exit handling, a default file protection mask, or if it is a child process that should inherit context.

Although many of the initialization activities are performed only once, DECC\$CRTL\_INIT can safely be called multiple times. On OpenVMS VAX systems, DECC\$CRTL\_INIT establishes the *Compaq C* RTL internal OpenVMS exception handler in the frame of the routine that calls DECC\$CRTL\_INIT each time DECC\$CRTL\_INIT is called.

At least one frame in the current call stack must have that handler established for OpenVMS exceptions to get mapped to UNIX signals.

---

## VAXC\$ESTABLISH

Used for establishing an OpenVMS exception handler for a particular routine. This function establishes a special *Compaq C* RTL exception handler in the routine that called it. This special handler catches all RTL-related exceptions that occur in later routines, and passes on all other exceptions to your handler.

### Format

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

```
void VAXC$ESTABLISH (unsigned int (*exception_handler)(void *sigarr, void *mecharr));
```

### Arguments

**exception\_handler**

The name of the function that you want to establish as an OpenVMS exception handler. You pass a pointer to this function as the parameter to VAXC\$ESTABLISH.

**sigarr**

A pointer to the signal array.

**mecharr**

A pointer to the mechanism array.

### Description

VAXC\$ESTABLISH must be used in place of LIB\$ESTABLISH when programs use the *Compaq C* RTL routines `setjmp` or `longjmp`. See `setjmp` and `longjmp`, or `sigsetjmp` and `siglongjmp` in this section.

You can only invoke the VAXC\$ESTABLISH function from a *Compaq C* for *OpenVMS* function, because it relies on the allocation of data space on the run-time stack by the *Compaq C* compiler. Calling the OpenVMS system library routine LIB\$ESTABLISH directly from a *Compaq C* function results in undefined behavior from the `setjmp` and `longjmp` functions.

To cause an OpenVMS exception to generate a UNIX style signal, user exception handlers must return `SS$_RESIGNAL` upon receiving any exception that they do not want to handle. Returning `SS$_NORMAL` prevents the generation of a UNIX style signal. UNIX signals are generated as if by an exception handler in the stack frame of the main C program. Not all OpenVMS exceptions correspond to UNIX signals. See Chapter 4 for more information on the interaction of OpenVMS exceptions and UNIX style signals.

Calling VAXC\$ESTABLISH with an argument of `NULL` cancels an existing handler in that routine.

---

### Notes

---

- On OpenVMS Alpha systems, VAXC\$ESTABLISH is implemented as a compiler built-in function, not as a *Compaq C* RTL function. (*Alpha only*)

## VAXC\$ESTABLISH

- On OpenVMS VAX systems, programs compiled with /NAMES=AS\_IS should link against SYS\$LIBRARY:DECCRTL.OLB to resolve the name VAXC\$ESTABLISH, whether or not the program is compiled with the /PREFIX\_LIBRARY\_ENTRIES switch. This is a restriction in the implementation. (*VAX only*)
-

---

## va\_arg

Returns the next item in the argument list.

### Format

```
#include <stdarg.h> (ANSI C)
#include <varargs.h> (COMPAQ C Extension)

type va_arg (va_list ap, type);
```

### Arguments

**ap**

A variable list containing the next argument to be obtained.

**type**

A data type that is used to determine the size of the next item in the list. An argument list can contain items of varying sizes, but the calling routine must determine what type of argument is expected since it cannot be determined at run time.

### Description

The `va_arg` function interprets the object at the address specified by the list incrementor according to type. If there is no corresponding argument, the behavior is undefined.

When using `va_arg` to write portable applications, include the `<stdarg.h>` header file (defined by the ANSI C standard), not the `<varargs.h>` header file, and use `va_arg` only in conjunction with other functions and macros defined in `<stdarg.h>`.

For an example of argument-list processing using the `<stdarg.h>` functions and definitions, see Example 3–6.

## va\_count

---

## va\_count

Returns the number of longwords (*VAX only*) or quadwords (*Alpha only*) in the argument list.

### Format

```
#include <stdarg.h> (ANSI C)
#include <varargs.h> (COMPAQ C Extension)
void va_count (int count);
```

### Argument

#### **count**

An integer variable name in which the number of longwords (*VAX only*) or quadwords (*Alpha only*) is returned.

### Description

The `va_count` macro places the number of longwords (*VAX only*) or quadwords (*Alpha only*) in the argument list into *count*. The value returned in *count* is the number of longwords (*VAX only*) or quadwords (*Alpha only*) in the function argument block not counting the *count* field itself.

If the argument list contains items whose storage requirements are a longword (*VAX only*) or quadword (*Alpha only*) of memory or less, the number in the *count* argument is also the number of arguments. However, if the argument list contains items that are longer than a longword (*VAX only*) or a quadword (*Alpha only*), *count* must be interpreted to obtain the number of arguments. Because a double is 8 bytes, it occupies two argument-list positions on OpenVMS VAX systems, and one argument-list position on OpenVMS Alpha systems.

The `va_count` macro is specific to *Compaq C* for *OpenVMS* Systems and is not portable.



---

## va\_end

Finishes the `<varargs.h>` or `<stdarg.h>` session.

### Format

```
#include <stdarg.h> (ANSI C)
#include <varargs.h> (COMPAQ C Extension)

void va_end (va_list ap);
```

### Argument

#### **ap**

The object used to traverse the argument list length. You must declare and use the argument *ap* as shown in this format section.

### Description

You can execute multiple traversals of the argument list, each delimited by `va_start ... va_end`. The `va_end` function sets *ap* equal to `NULL`.

When using this function to write portable applications, include the `<stdarg.h>` header file (defined by the ANSI C standard), not the `<varargs.h>` header file, and use `va_end` only in conjunction with other routines defined in `<stdarg.h>`.

For an example of argument-list processing using the `<stdarg.h>` functions and definitions, see Example 3–6.

## va\_start\_1, va\_start

---

## va\_start\_1, va\_start

Used for initializing a variable to the beginning of the argument list.

### Format

```
#include <varargs.h> (COMPAQ C Extension)

void va_start (va_list ap);

void va_start_1 (va_list ap, int offset);
```

### Arguments

#### ap

An object pointer. You must declare and use the argument *ap* as shown in the format section.

#### offset

The number of bytes by which *ap* is to be incremented so that it points to a subsequent argument within the list (that is, not to the start of the argument list). Using a nonzero offset can initialize *ap* to the address of the first of the optional arguments that follow a number of fixed arguments.

### Description

The `va_start` macro initializes the variable *ap* to the beginning of the argument list.

The `va_start_1` macro initializes *ap* to the address of an argument that is preceded by a known number of defined arguments. The `printf` function is an example of a *Compaq C* RTL function that contains a variable-length argument list offset from the beginning of the entire argument list. The variable-length argument list is offset by the address of the formatting string.

When determining value of the offset argument used in `va_start_1` the implications of the OpenVMS calling standard must be considered.

On OpenVMS VAX, most argument items are a longword. For example, OpenVMS VAX arguments of types `char` and `short` use a full longword of memory when they are present in argument lists. However, OpenVMS VAX arguments of type `float` use two longwords because they are converted to type `double`.

On OpenVMS Alpha, each argument item is a quadword.

---

#### Note

When accessing argument lists, especially those passed to a subroutine (written in C) by a program written in another programming language, consider the implications of the OpenVMS calling standard. For more information about the OpenVMS calling standard, see the *Compaq C User's Guide for OpenVMS Systems* or the *OpenVMS Calling Standard*.

---

The preceding version of `va_start` and `va_start_1` is specific to the *Compaq C* RTL, and is not portable.

The following syntax describes the `va_start` macro in the `<stdarg.h>` header file, as defined in the ANSI C standard:

## Format

```
#include <stdarg.h> (ANSI C)

void va_start (va_list ap, parmN);
```

## Arguments

### **ap**

An object pointer. You must declare and use the argument *ap* as shown in the format section.

### **parmN**

The name of the last of the known fixed arguments.

## Description

The pointer *ap* is initialized to point to the first of the optional arguments that follow *parmN* in the argument list.

Always use this version of `va_start` in conjunction with functions that are declared and defined with function prototypes. Also use this version of `va_start` to write portable programs.

For an example of argument-list processing using the `<stdarg.h>` functions and definitions, see Example 3–6.

---

## vfork

Creates an independent child process. This function is nonreentrant.

### Format

```
#include <unistd.h>

int vfork (void); (_DECC_V4_SOURCE)

pid_t vfork (void); (not _DECC_V4_SOURCE)
```

### Description

The `vfork` function provided by *Compaq C* for *OpenVMS* Systems differs from the `fork` function provided by other C implementations. Table REF–12 shows the two major differences.

**Table REF–12 The `vfork` and `fork` Functions**

| The <code>vfork</code> Function                                                        | The <code>fork</code> Function                                                                                                                                                                        |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Used with the <code>exec</code> functions.                                             | Can be used without an <code>exec</code> function for asynchronous processing.                                                                                                                        |
| Creates an independent child process that shares some of the parent's characteristics. | Creates an exact duplicate of the parent process that branches at the point where <code>vfork</code> is called, as if the parent and the child are the same process at different stages of execution. |

The `vfork` function provides the setup necessary for a subsequent call to an `exec` function. Although no process is created by `vfork`, it performs the following steps:

- It saves the return address (the address of the `vfork` call) to be used later as the return address for the call to an `exec` function.
- It saves the current context.
- It returns the integer 0 the first time it is called (before the call to an `exec` function is made). After the corresponding `exec` function call is made, the `exec` function returns control to the parent process, at the point of the `vfork` call, and it returns the process ID of the child as the return value. Unless the `exec` function fails, control appears to return twice from `vfork` even though one call was made to `vfork` and one call was made to the `exec` function.

The behavior of the `vfork` function is similar to the behavior of the `setjmp` function. Both `vfork` and `setjmp` establish a return address for later use, both return the integer 0 when they are first called to set up this address, and both pass back the second return value as though it were returned by them rather than by their corresponding `exec` or `longjmp` function calls.

However, unlike `setjmp`, with `vfork`, all local automatic variables, even those with volatile-qualified type, can have indeterminate values if they are modified between the call to `vfork` and the corresponding call to an `exec` routine.

**Return Values**

|         |                                                           |
|---------|-----------------------------------------------------------|
| 0       | Indicates successful creation of the context.             |
| nonzero | Indicates the process ID (PID) of the child process.      |
| -1      | Indicates an error – failure to create the child process. |

## fprintf

---

## fprintf

Prints formatted output based on an argument list.

### Format

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

```
int fprintf (FILE *file_ptr, const char *format, va_list arg);
```

### Arguments

**file\_ptr**

A pointer to the file to which the output is directed.

**format**

A pointer to a string containing the format specification. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

**arg**

A list of expressions whose resultant types correspond to the conversion specifications given in the format specifications.

### Description

See also `vprintf` and `vsprintf` in this section.

### Return Values

x

The number of bytes written.

Negative value

Indicates an output error. The function sets `errno`. For a list of possible `errno` values set, see `fprintf` in this section.

---

## vfprintf

Reads formatted input based on an argument list.

### Format

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

```
int vfprintf (FILE *file_ptr, const char *format, va_list arg);
```

### Arguments

**file\_ptr**

A pointer to the file that provides input text.

**format**

A pointer to a string containing the format specification.

**arg**

A list of expressions whose resultant types correspond to the conversion specifications given in the format specifications.

### Description

The vfprintf function is the same as the fprintf function except that instead of being called with a variable number of arguments, it is called with an argument list that has been initialized by va\_start (and possibly subsequent va\_arg calls).

If no conversion specifications are given, you can omit the input pointers. Otherwise, the function calls must have exactly as many input pointers as there are conversion specifications, and the conversion specifications must match the types of the input pointers.

Conversion specifications are matched to input sources in left-to-right order. Excess input pointers, if any, are ignored.

For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

This function returns the number of successfully matched and assigned input items.

See also the fscanf and vscanf functions in this section.

### Return Values

n

The number of successfully matched and assigned input items.

EOF

Indicates that the end-of-file was encountered or a read error occurred. If a read error occurs, the function sets `errno` to one of the following:

- `EILSEQ` – Invalid character detected.
- `EINVAL` – Insufficient arguments.
- `ENOMEM` – Not enough memory available for conversion.
- `ERANGE` – Floating-point calculations overflow.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This can indicate that conversion to a numeric value failed due to overflow.

The function can also set `errno` to the following as a result of errors returned from the I/O subsystem:

- `EBADF` – The file descriptor is not valid.
- `EIO` – I/O error.
- `ENXIO` – Device does not exist.
- `EPIPE` – Broken pipe.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This indicates that an I/O error occurred for which there is no equivalent C error code.



---

## vfwprintf

Writes output to the stream under control of the wide-character format string.

### Format

```
#include <wchar.h>

int vfwprintf (FILE *stream, const wchar_t *format, va_list arg);
```

### Arguments

**stream**

A file pointer.

**format**

A pointer to a wide-character string containing the format specifications. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

**arg**

A variable list of the items needed for output.

### Description

The `vfwprintf` function is equivalent to the `fwprintf` function, with the variable argument list replaced by the *arg* argument. Initialize *arg* with the `va_start` macro (and possibly with subsequent `va_arg` calls) from `<stdarg.h>`.

If the stream pointed to by *stream* has no orientation, `vfwprintf` makes the stream wide-oriented.

See also `fwprintf` in this section.

### Return Values

|   |                                        |
|---|----------------------------------------|
| n | The number of wide characters written. |
|---|----------------------------------------|

Negative value

Indicates an error. The function sets `errno` to one of the following:

- `EILSEQ` – Invalid character detected.
- `EINVAL` – Insufficient arguments.
- `ENOMEM` – Not enough memory available for conversion.
- `ERANGE` – Floating-point calculations overflow.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This might indicate that conversion to a numeric value failed because of overflow.

The function can also set `errno` to the following as a result of errors returned from the I/O subsystem:

- `EBADF` – The file descriptor is not valid.
- `EIO` – I/O error.
- `ENOSPC` – No free space on the device containing the file.
- `ENXIO` – Device does not exist.
- `EPIPE` – Broken pipe.
- `ESPIPE` – Illegal seek in a file opened for append.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This indicates that an I/O error occurred for which there is no equivalent C error code.

## Examples

The following example shows the use of the `vfwprintf` function in a general error reporting routine.

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

void error(char *function_name, wchar_t *format, ...);
{
 va_list args;
 va_start(args, format);
 /* print out name of function causing error */
 fwprintf(stderr, L"ERROR in %s: ", function_name);
 /* print out remainder of message */
 vfwprintf(stderr, format, args);
 va_end(args);
}
```

---

## vfwscanf

Reads input from the stream under control of a wide-character format string.

### Format

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

```
int vfwscanf (FILE *stream, const wchar_t *format, va_list arg);
```

### Arguments

**stream**

A file pointer.

**format**

A pointer to a wide-character string containing the format specifications.

**arg**

A list of expressions whose resultant types correspond to the conversion specifications given in the format specifications.

### Description

The `vfwscanf` function is equivalent to the `fwscanf` function, except that instead of being called with a variable number of arguments, it is called with an argument list (*arg*) that has been initialized by `va_start` (and possibly with subsequent `va_arg` calls).

If the stream pointed to by *stream* has no orientation, `vfwscanf` makes the stream wide-oriented.

For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

### Return Values

|     |                                                                                                                                                                                             |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n   | The number of successfully matched and assigned wide-character input items.                                                                                                                 |
| EOF | Indicates that a read error occurred before any conversion. The function sets <code>errno</code> . For a list of the values set by this function, see <code>vfscanf</code> in this section. |

## vprintf

---

## vprintf

Prints formatted output based on an argument list.

This function is the same as the `printf` function except that instead of being called with a variable number of arguments, it is called with an argument list that has been initialized by the `va_start` macro (and possibly with subsequent `va_arg` calls) from `<stdarg.h>`.

### Format

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

```
int vprintf (const char *format, va_list arg);
```

### Arguments

#### **format**

A pointer to the string containing the format specification. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

#### **arg**

A variable list of the items needed for output.

### Description

See the `vfprintf` and `vsprintf` functions this section.

### Return Values

|                |                                                                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x              | The number of bytes written.                                                                                                                                      |
| Negative value | Indicates an output error. The function sets <code>errno</code> . For a list of possible <code>errno</code> values set, see <code>fprintf</code> in this section. |

---

## vscanf

Reads formatted input based on an argument list.

### Format

```
#include <stdio.h>

int vscanf (const char *format, va_list arg);
```

### Arguments

**format**

A pointer to the string containing the format specification.

**arg**

A list of expressions whose resultant types correspond to the conversion specifications given in the format specifications.

### Description

The `vscanf` function is the same as the `scanf` function except that instead of being called with a variable number of arguments, it is called with an argument list (*arg*) that has been initialized by the `va_start` macro (and possibly with subsequent `va_arg` calls).

For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

See the `scanf`, `vfscanf`, and `vsscanf` functions this section.

### Return Values

|     |                                                                                                                                                                                             |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n   | The number of successfully matched and assigned input items.                                                                                                                                |
| EOF | Indicates that a read error occurred before any conversion. The function sets <code>errno</code> . For a list of the values set by this function, see <code>vfscanf</code> in this section. |

## vsprintf

---

## vsprintf

Prints formatted output based on an argument list.

This function is the same as the `sprintf` function except that instead of being called with a variable number of arguments, it is called with an argument list that has been initialized by `va_start` (and possibly with subsequent `va_arg` calls).

### Format

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

```
int vsprintf (char *str, const char *format, va_list arg);
```

### Arguments

#### **str**

A pointer to a string that will receive the formatted output. This string is assumed to be large enough to hold the output.

#### **format**

A pointer to a character string that contains the format specification. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

#### **arg**

A list of expressions whose resultant types correspond to the conversion specifications given in the format specifications.

### Return Value

x

The number of bytes written.

Negative value

Indicates an output error occurred. The function sets `errno`. For a list of possible `errno` values set, see `fprintf` in this section.

---

## vsscanf

Reads formatted input based on an argument list.

### Format

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

```
int vsscanf (char *str, const char *format, va_list arg);
```

### Arguments

**str**

The address of the character string that provides the input text to sscanf.

**format**

A pointer to a character string that contains the format specification.

**arg**

A list of expressions whose resultant types correspond to the conversion specifications given in the format specifications.

### Description

The vsscanf function is the same as the sscanf function except that instead of being called with a variable number of arguments, it is called with an argument list that has been initialized by va\_start (and possibly with subsequent va\_arg calls).

The vsscanf function is also equivalent to the vfscanf function, except that the first argument specifies a wide-character string rather than a stream. Reaching the end of the wide-character string is the same as encountering EOF for the vfscanf function.

For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

See the vsscanf and sscanf functions in this section.

### Return Values

|     |                                                                                                                                                                  |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n   | The number of successfully matched and assigned input items.                                                                                                     |
| EOF | Indicates that a read error occurred before any conversion. The function sets errno. For a list of the values set by this function, see vfscanf in this section. |

## vswprintf

---

## vswprintf

Writes output to the stream under control of the wide-character format string.

### Format

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

```
int vswprintf (wchar_t *s, size_t n, const wchar_t *format, va_list arg);
```

### Arguments

**s**

A pointer to a multibyte character sequence.

**n**

The maximum number of bytes that comprise the multibyte character.

**format**

A pointer to a wide-character string containing the format specifications. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

**arg**

A variable list of the items needed for output.

### Description

The `vswprintf` function is equivalent to the `swprintf` function, with the variable argument list replaced by the *arg* argument. Initialize *arg* with the `va_start` macro, and possibly with subsequent `va_arg` calls.

See the `swprintf` function in this section.

### Return Values

*n*

The number of wide characters written.



Negative value

Indicates an error. The function sets `errno` to one of the following:

- `EILSEQ` – Invalid character detected.
- `EINVAL` – Insufficient arguments.
- `ENOMEM` – Not enough memory available for conversion.
- `ERANGE` – Floating-point calculations overflow.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This might indicate that conversion to a numeric value failed because of overflow.

The function can also set `errno` to the following as a result of errors returned from the I/O subsystem:

- `EBADF` – The file descriptor is not valid.
- `EIO` – I/O error.
- `ENOSPC` – No free space on the device containing the file.
- `ENXIO` – Device does not exist.
- `EPIPE` – Broken pipe.
- `ESPIPE` – Illegal seek in a file opened for append.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This indicates that an I/O error occurred for which there is no equivalent C error code.

---

## vswscanf

Reads input from the stream under control of the wide-character format string.

### Format

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

```
int vswscanf (wchar_t *s, const wchar_t *format, va_list arg);
```

### Arguments

#### **s**

A pointer to a wide-character string from which the input is to be obtained.

#### **format**

A pointer to a wide-character string containing the format specifications.

#### **arg**

A list of expressions whose results correspond to conversion specifications given in the format specification.

### Description

The `vswscanf` function is equivalent to the `swscanf` function, except that instead of being called with a variable number of arguments, it is called with an argument list (*arg*) that has been initialized by `va_start` (and possibly with subsequent `va_arg` calls).

The `vswscanf` function is also equivalent to the `vfwscanf` function, except that the first argument specifies a wide-character string rather than a stream. Reaching the end of the wide-character string is the same as encountering EOF for the `vfwscanf` function.

For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

See the `vfwscanf` and `swscanf` functions in this section.

### Return Values

`n`

The number of wide characters read.

EOF

Indicates that a read error occurred before any conversion. The function sets `errno`. For a list of the values set by this function, see `vfscanf` in this section.

---

## vwprintf

Writes output to an array of wide characters under control of the wide-character format string.

### Format

```
#include <wchar.h>

int vwprintf (const wchar_t *format, va_list arg);
```

### Arguments

**format**

A pointer to a wide-character string containing the format specifications. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

**arg**

The variable list of items needed for output.

### Description

The `vwprintf` function is equivalent to the `wprintf` function, with the variable argument list replaced by the *arg* argument. Initialize *arg* with the `va_start` macro, and possibly with subsequent `va_arg` calls. The `vwprintf` function does not invoke the `va_end` macro.

See also `wprintf` in this section.

### Return Values

|                |                                                                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x              | The number of wide characters written, not counting the terminating null wide character.                                                                                                   |
| Negative value | Indicates an error. Either <i>n</i> or more wide characters were requested to be written, or a conversion error occurred, in which case <code>errno</code> is set to <code>EILSEQ</code> . |

---

## vwscanf

Reads input from an array of wide characters under control of a wide-character format string.

### Format

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

```
int vwscanf (const wchar_t *format, va_list arg);
```

### Arguments

#### **format**

A pointer to a wide-character string containing the format specifications.

#### **arg**

A list of expressions whose resultant types correspond to the conversion specifications given in the format specifications.

### Description

The `vwscanf` function is equivalent to the `wscanf` function, except that instead of being called with a variable number of arguments, it is called with an argument list (*arg*) that has been initialized by `va_start` (and possibly with subsequent `va_arg` calls).

For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

See also the `wscanf` function in this section.

### Return Values

`n`

The number of wide characters read.

EOF

Indicates that a read error occurred before any conversion. The function sets `errno`. For a list of the values set by this function, see `vfscanf` in this section.

---

## wait

Checks the status of the child process before exiting. A child process is terminated when the parent process terminates.

### Format

```
#include <wait.h>

pid_t wait (int *status);
```

### Argument

#### **status**

The address of a location to receive the final status of the terminated child. The child can set the status with the `exit` function and the parent can retrieve this value by specifying *status*.

### Description

The `wait` function suspends the parent process until the final status of a terminated child is returned from the child.

On OpenVMS Version 7.0 and higher systems, the `wait` function is equivalent to `waitpid( 0, status, 0 )` if you include `<wait.h>` and compile with the `_POSIX_EXIT` feature-test macro set (either with `/DEFINE=_POSIX_EXIT` or with `#define _POSIX_EXIT` at the top of your file, before any file inclusions).

### Return Values

|    |                                                                                                                                                                                                                                                                                        |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | The process ID (PID) of the terminated child. If more than one child process was created, <code>wait</code> will return the PID of the terminated child that was most recently created. Subsequent calls will return the PID of the next most recently created, but terminated, child. |
| -1 | No child process was spawned.                                                                                                                                                                                                                                                          |

---

**wait3**

Waits for a child process to stop or terminate.

**Format**

```
#include <wait.h>
```

```
pid_t wait3 (int *status_location, int options, struct rusage *resource_usage);
```

**Arguments****status\_location**

A pointer to a location that contains the termination status of the child process as defined in the `<wait.h>` header file.

Beginning with OpenVMS Version 7.2, when compiled with the `_VMS_WAIT` macro defined, the `wait3` function puts the OpenVMS completion code of the child process at the address specified in the *status\_location* argument.

**options**

Flags that modify the behavior of the function. These flags are defined in the Description section.

**resource\_usage**

The location of a structure that contains the resource utilization information for terminated child processes.

**Description**

The `wait3` function suspends the calling process until the request is completed, and redefines it so that only the calling thread is suspended.

The *options* argument modifies the behavior of the function. You can combine the flags for the *options* argument by specifying their bitwise inclusive OR. The flags are:

|           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WNOWAIT   | Specifies that the process whose status is returned in <i>status_location</i> is kept in a waitable state. You can wait for the process again with the same results.                                                                                                                                                                                                                                                                                                                                       |
| WNOHANG   | Prevents the suspension of the calling process. If there are child processes that stopped or terminated, one is chosen and the <code>waitpid</code> function returns its process ID, as when you do not specify the <code>WNOHANG</code> flag. If there are no terminated processes (that is, if <code>waitpid</code> suspends the calling process without the <code>WNOHANG</code> flag), 0 (zero) is returned. Because you can never wait for process 0, there is no confusion arising from this return. |
| WUNTRACED | Specifies that the call return additional information when the child processes of the current process stop because the child process received a <code>SIGTTIN</code> , <code>SIGTOU</code> , <code>SIGSTOP</code> , or <code>SIGTSTOP</code> signal.                                                                                                                                                                                                                                                       |

If the `wait3` function returns because the status of a child process is available, the process ID of the child process is returned. Information is stored in the location pointed to by *status\_location*, if this pointer is not null.

The value stored in the location pointed to by *status\_location* is 0 (zero) only if the status is returned from a terminated child process that did one of the following:

- Returned 0 from the main function.
- Passed 0 as the *status* argument to the `_exit` or `exit` function.

Regardless of the *status\_location* value, you can define this information using the macros defined in the `<wait.h>` header file, which evaluate to integral expressions. In the following macro descriptions, the *status\_value* argument is equal to the integer value pointed to by the *status\_location* argument:

|                                         |                                                                                                                                                                                                                                                                                                       |
|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>WIFEXITED(status_value)</code>    | Evaluates to a nonzero value if status was returned for a child process that terminated normally.                                                                                                                                                                                                     |
| <code>WEXITSTATUS(status_value)</code>  | If the value of <code>WIFEXITED(status_value)</code> is nonzero, this macro evaluates to the low-order 8 bits of the <i>status</i> argument that the child process passed to the <code>_exit</code> or <code>exit</code> function, or to the value the child process returned from the main function. |
| <code>WIFSIGNALED(status_value)</code>  | Evaluates to nonzero value if status was returned for a child process that terminated due to the receipt of a signal that was not caught.                                                                                                                                                             |
| <code>WTERMSIG(status_value)</code>     | If the value of <code>WIFSIGNALED(status_value)</code> is nonzero, this macro evaluates to the number of the signal that caused the termination of the child process.                                                                                                                                 |
| <code>WIFSTOPPED(status_value)</code>   | Evaluates to a nonzero value if status was returned for a child process that is currently stopped.                                                                                                                                                                                                    |
| <code>WSTOPSIG(status_value)</code>     | If the value of <code>WIFSTOPPED(status_value)</code> is nonzero, this macro evaluates to the number of the signal that caused the child process to stop.                                                                                                                                             |
| <code>WIFCONTINUED(status_value)</code> | Evaluates to a nonzero value if status was returned for a child process that has continued.                                                                                                                                                                                                           |

If the information stored at the location pointed to by *status\_location* was stored there by a call to `wait3` that specified the `WUNTRACED` flag, one of the following macros evaluates to a nonzero value:

- `WIFEXITED(*status_value)`
- `WIFSIGNALED(*status_value)`
- `WIFSTOPPED(*status_value)`
- `WIFCONTINUED(*status_value)`

If the information stored in the location pointed to by *status\_location* resulted from a call to `wait3` without the `WUNTRACED` flag specified, one of the following macros evaluates to a nonzero value:

- `WIFEXITED(*status_value)`
- `WIFSIGNALED(*status_value)`

## wait3

The `wait3` function provides compatibility with BSD systems. The *resource\_usage* argument points to a location that contains resource usage information for the child processes as defined in the `<resource.h>` header file.

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes is assigned a parent process ID equal to the process ID of the `init` process.

See also `exit`, `-exit`, and `init` in this section.

### Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success. There are no stopped or exited child processes, the <code>WNOHANG</code> option is specified.                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| x  | The <i>process_id</i> of the child process. Status of a child process is available.                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| -1 | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"><li>• <code>ECHILD</code> – There are no child processes to wait for.</li><li>• <code>EINTR</code> – Terminated by receipt of a signal caught by the calling process.</li><li>• <code>EFAULT</code> – The <i>status_location</i> or <i>resource_usage</i> argument points to a location outside of the address space of the process.</li><li>• <code>EINVAL</code>— The value of the <i>options</i> argument is not valid.</li></ul> |



---

## wait4

Waits for a child process to stop or terminate.

### Format

```
#include <wait.h>
```

```
pid_t wait4 (pid_t process_id, union wait *status_location, int options, struct rusage *resource_usage);
```

### Arguments

#### **status\_location**

A pointer to a location that contains the termination status of the child process as defined in the `<wait.h>` header file.

Beginning with OpenVMS Version 7.2, when compiled with the `_VMS_WAIT` macro defined, the `wait4` function puts the OpenVMS completion code of the child process at the address specified in the *status\_location* argument.

#### **process\_id**

The child process or set of child processes.

#### **options**

Flags that modify the behavior of the function. These flags are defined in the Description section.

#### **resource\_usage**

The location of a structure that contains the resource utilization information for terminated child processes.

### Description

The `wait4` function suspends the calling process until the request is completed.

The *process\_id* argument allows the calling process to gather status from a specific set of child processes, according to the following rules:

| If the <i>process_id</i> is | Then status is requested                                                                                                   |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Equal to <code>-1</code>    | For any child process. In this respect, the <code>waitpid</code> function is equivalent to the <code>wait</code> function. |
| Greater than <code>0</code> | For a single child process and specifies the process ID.                                                                   |

The `wait4` function only returns the status of a child process from this set.

The *options* argument to the `wait4` function modifies the behavior of the function. You can combine the flags for the *options* argument by specifying their bitwise-inclusive OR. The flags are:

#### **WNOWAIT**

Specifies that the process whose status is returned in *status\_location* is kept in a waitable state. You can wait for the process again with the same results.

## wait4

|           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WNOHANG   | Prevents the suspension of the calling process. If there are child processes that stopped or terminated, one is chosen and the <code>waitpid</code> function returns its process ID, as when you do not specify the WNOHANG flag. If there are no terminated processes (that is, if <code>waitpid</code> suspends the calling process without the WNOHANG flag), 0 is returned. Because you can never wait for process 0, there is no confusion arising from this return. |
| WUNTRACED | Specifies that the call return additional information when the child processes of the current process stop because the child process received a SIGTTIN, SIGTTOU, SIGSTOP, or SIGTSTOP signal.                                                                                                                                                                                                                                                                            |

If the `wait4` function returns because the status of a child process is available, the process ID of the child process is returned. Information is stored in the location pointed to by *status\_location*, if this pointer is not null.

The value stored in the location pointed to by *status\_location* is 0 only if the status is returned from a terminated child process that did one of the following:

- Returned 0 from the main function.
- Passed 0 as the *status* argument to the `_exit` or `exit` function.

Regardless of the *status\_location* value, you can define this information using the macros defined in the `<wait.h>` header file, which evaluate to integral expressions. In the following macro descriptions, *status\_value* is equal to the integer value pointed to by *status\_location*:

|                                    |                                                                                                                                                                                                                                                                                                   |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WIFEXITED( <i>status_value</i> )   | Evaluates to a nonzero value if status was returned for a child process that terminated normally.                                                                                                                                                                                                 |
| WEXITSTATUS( <i>status_value</i> ) | If the value of WIFEXITED( <i>status_value</i> ) is nonzero, this macro evaluates to the low-order 8 bits of the <i>status</i> argument that the child process passed to the <code>_exit</code> or <code>exit</code> function, or to the value the child process returned from the main function. |
| WIFSIGNALED( <i>status_value</i> ) | Evaluates to nonzero value if status was returned for a child process that terminated due to the receipt of a signal that was not caught.                                                                                                                                                         |
| WTERMSIG( <i>status_value</i> )    | If the value of WIFSIGNALED( <i>status_value</i> ) is nonzero, this macro evaluates to the number of the signal that caused the termination of the child process.                                                                                                                                 |
| WIFSTOPPED( <i>status_value</i> )  | Evaluates to a nonzero value if status was returned for a child process that is currently stopped.                                                                                                                                                                                                |
| WSTOPSIG( <i>status_value</i> )    | If the value of WIFSTOPPED( <i>status_value</i> ) is nonzero, this macro evaluates to the number of the signal that caused the child process to stop.                                                                                                                                             |

`WIFCONTINUED(status_value)` Evaluates to a nonzero value if status was returned for a child process that has continued.

If the information stored at the location pointed to by *status\_location* was stored there by a call to `wait4` that specified the `WUNTRACED` flag, one of the following macros evaluates to a nonzero value:

- `WIFEXITED(*status_value)`
- `WIFSIGNALED(*status_value)`
- `WIFSTOPPED(*status_value)`
- `WIFCONTINUED(*status_value)`

If the information stored in the location pointed to by *status\_location* resulted from a call to `wait4` without the `WUNTRACED` flag specified, one of the following macros evaluates to a nonzero value:

- `WIFEXITED(*status_value)`
- `WIFSIGNALED(*status_value)`

The `wait4` function is similar to the `wait3` function. However, the `wait4` function waits for a specific child as indicated by the *process\_id* argument. The *resource\_usage* argument points to a location that contains resource usage information for the child processes as defined in the `<resource.h>` header file.

See also `exit` and `_exit` in this section.

## Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success. There are no stopped or exited child processes, the <code>WNOHANG</code> option is specified.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| x  | The <i>process_id</i> of the child process. Status of a child process is available.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| -1 | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"> <li>• <code>ECHILD</code> – There are no child processes to wait for.</li> <li>• <code>EINTR</code> – Terminated by receipt of a signal caught by the calling process.</li> <li>• <code>EFAULT</code> – The <i>status_location</i> or <i>resource_usage</i> argument points to a location outside of the address space of the process.</li> <li>• <code>EINVAL</code>— The value of the <i>options</i> argument is not valid.</li> </ul> |

---

## waitpid

Waits for a child process to stop or terminate.

### Format

```
#include <wait.h>

pid_t waitpid (pid_t process_id, int *status_location, int options);
```

### Arguments

#### **process\_id**

The child process or set of child processes.

#### **status\_location**

A pointer to a location that contains the termination status of the child process as defined in the `<wait.h>` header file.

Beginning with OpenVMS Version 7.2, when compiled with the `_VMS_WAIT` macro defined, the `waitpid` function puts the OpenVMS completion code of the child process at the address specified in the *status\_location* argument.

#### **options**

Flags that modify the behavior of the function. These flags are defined in the Description section.

### Description

The `waitpid` function suspends the calling process until the request is completed. It is redefined so that only the calling thread is suspended.

If the *process\_id* argument is `-1` and the *options* argument is `0`, the `waitpid` function behaves the same as the `wait` function. If these arguments have other values, the `waitpid` function is changed as specified by those values.

The *process\_id* argument allows the calling process to gather status from a specific set of child processes, according to the following rules:

| If the <i>process_id</i> is | Then status is requested                                                                                                   |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------|
| Equal to <code>-1</code>    | For any child process. In this respect, the <code>waitpid</code> function is equivalent to the <code>wait</code> function. |
| Greater than <code>0</code> | For a single child process and specifies the process ID.                                                                   |

The `waitpid` function only returns the status of a child process from this set.

The *options* argument to the `waitpid` function modifies the behavior of the function. You can combine the flags for the *options* argument by specifying their bitwise-inclusive OR. The flags are:

|            |                                                                                                                                                                                                       |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WCONTINUED | Specifies that the following is reported to the calling process: the status of any continued child process specified by the <i>process_id</i> argument whose status is unreported since it continued. |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|           |                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WNOHANG   | Specifies that the process whose status is returned in <i>status_location</i> is kept in a waitable state. You can wait for the process again with the same results.                                                                                                                                                                                                                                                                  |
| WUNTRACED | Prevents the calling process from being suspended. If there are child processes that stopped or terminated, one is chosen and waitpid returns its pid, as when you do not specify the WNOHANG flag. If there are no terminated processes (that is, if waitpid suspends the calling process without the WNOHANG flag), 0 (zero) is returned. Because you can never wait for process 0, there is no confusion arising from this return. |
| WNOHANG   | Specifies that the call return additional information when the child processes of the current process stop because the child process received a SIGTTIN, SIGTTOU, SIGSTOP, or SIGTSTOP signal.                                                                                                                                                                                                                                        |

If the waitpid function returns because the status of a child process is available, the process ID of the child process is returned. Information is stored in the location pointed to by *status\_location*, if this pointer is not null. The value stored in the location pointed to by *status\_location* is 0 only if the status is returned from a terminated child process that did one of the following:

- Returned 0 from the main function.
- Passed 0 as the *status* argument to the `_exit` or `exit` function.

Regardless of the value of *status\_location*, you can define this information using the macros defined in the `<wait.h>` header file, which evaluate to integral expressions. In the following function descriptions, *status\_value* is equal to the integer value pointed to by *status\_location*:

|                                    |                                                                                                                                                                                                                                                                                                   |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WIFEXITED( <i>status_value</i> )   | Evaluates to a nonzero value if status was returned for a child process that terminated normally.                                                                                                                                                                                                 |
| WEXITSTATUS( <i>status_value</i> ) | If the value of WIFEXITED( <i>status_value</i> ) is nonzero, this macro evaluates to the low-order 8 bits of the <i>status</i> argument that the child process passed to the <code>_exit</code> or <code>exit</code> function, or to the value the child process returned from the main function. |
| WIFSIGNALED( <i>status_value</i> ) | Evaluates to nonzero value if status returned for a child process that terminated due to the receipt of a signal not caught.                                                                                                                                                                      |
| WTERMSIG( <i>status_value</i> )    | If the value of WIFSIGNALED( <i>status_value</i> ) is nonzero, this macro evaluates to the number of the signal that caused the termination of the child process.                                                                                                                                 |
| WIFSTOPPED( <i>status_value</i> )  | Evaluates to a nonzero value if status was returned for a child process that is currently stopped.                                                                                                                                                                                                |

## waitpid

|                                         |                                                                                                                                                           |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>WSTOPSIG(status_value)</code>     | If the value of <code>WIFSTOPPED(status_value)</code> is nonzero, this macro evaluates to the number of the signal that caused the child process to stop. |
| <code>WIFCONTINUED(status_value)</code> | Evaluates to a nonzero value if status returned for a child process that continued.                                                                       |

If the information stored at the location pointed to by *status\_location* is stored there by a call to `waitpid` that specified the `WUNTRACED` flag, one of the following macros evaluates to a nonzero value:

- `WIFEXITED(*status_value)`
- `WIFSIGNALED(*status_value)`
- `WIFSTOPPED(*status_value)`
- `WIFCONTINUED(*status_value)`

If the information stored in the buffer pointed to by *status\_location* resulted from a call to `waitpid` without the `WUNTRACED` flag specified, one of the following macros evaluates to a nonzero value:

- `WIFEXITED(*status_value)`
- `WIFSIGNALED(*status_value)`

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes is assigned a parent process ID equal to the process ID of the init process.

See also `exit`, `_exit`, and `wait` in this section.

## Return Values

|   |                                                                                                                                                                                       |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | Indicates success. If the <code>WNOHANG</code> option was specified, and there are no stopped or exited child processes, the <code>waitpid</code> function also returns a value of 0. |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

-1

Indicates an error; `errno` is set to one of the following values:

- **ECHILD**—The calling process has no existing unwaited-for child processes. The process or process group ID specified by the *process\_id* argument does not exist or is not a child process of the calling process.
- **EINTR**—The function was terminated by receipt of a signal.  
If the `waitpid` function returns because the status of a child process is available, the process ID of the child is returned to the calling process. If they return because a signal was caught by the calling process, `-1` is returned.
- **EFAULT**— The *status\_location* argument points to a location outside of the address space of the process.
- **EINVAL**— The value of the *options* argument is not valid.

---

## wcrtomb

Converts the wide character to its multibyte character representation.

### Format

```
#include <wchar.h>

size_t wcrtomb (char *s, wchar_t wc, mbstate_t *ps);
```

### Arguments

#### **s**

A pointer to the resulting multibyte character.

#### **wc**

A wide character.

#### **ps**

A pointer to the `mbstate_t` object. If a NULL pointer is specified, the function uses its internal `mbstate_t` object. `mbstate_t` is an opaque datatype intended to keep the conversion state for the state-dependent codesets.

### Description

If *s* is a NULL pointer, the `wcrtomb` function is equivalent to the call:

```
wcrtomb (buf, L'\0', ps)
```

where *buf* is an internal buffer.

If *s* is not a NULL pointer, the `wcrtomb` function determines the number of bytes needed to represent the multibyte character that corresponds to the wide character specified by *wc* (including any shift sequences), and stores the resulting bytes in the array whose first element is pointed to by *s*. At most `MB_CUR_MAX` bytes are stored.

If *wc* is a null wide character, a null byte is stored preceded by any shift sequence needed to restore the initial shift state. The resulting state described is the initial conversion state.

### Return Values

|                 |                                                                                                                                                                                     |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>n</code>  | The number of bytes stored in the resulting array, including any shift sequences to represent the multibyte character.                                                              |
| <code>-1</code> | Indicates an encoding error. The <i>wc</i> argument is not a valid wide character. The global <code>errno</code> is set to <code>EILSEQ</code> ; the conversion state is undefined. |



---

## wcscat

Concatenates two wide-character strings.

### Format

```
#include <wchar.h>

wchar_t *wcscat (wchar_t *wstr_1, const wchar_t *wstr_2);
```

### Function Variants

The `wcscat` function has variants named `_wcscat32` and `_wcscat64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**wstr\_1, wstr\_2**

Pointers to null-terminated wide-character strings.

### Description

The `wcscat` function appends the wide-character string `wstr_2`, including the terminating null character, to the end of `wstr_1`.

See also `wcsncat` in this section.

### Return Value

|   |                                                                                                                |
|---|----------------------------------------------------------------------------------------------------------------|
| x | The first argument, <code>wstr_1</code> , which is assumed to be large enough to hold the concatenated result. |
|---|----------------------------------------------------------------------------------------------------------------|

### Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

/* This program concatenates two wide-character strings using */
/* the wcscat function, and then manually compares the result */
/* to the expected result */

#define S1LENGTH 10
#define S2LENGTH 8

main()
{
 int i;
 wchar_t s1buf[S1LENGTH + S2LENGTH];
 wchar_t s2buf[S2LENGTH];
 wchar_t test1[S1LENGTH + S2LENGTH];

 /* Initialize the three wide-character strings */

 if (mbstowcs(s1buf, "abcmnxyz", S1LENGTH) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }
}
```

## wcscat

```
 if (mbstowcs(s2buf, " orthis", S2LENGTH) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 if (mbstowcs(test1, "abcmnexyz orthis", S1LENGTH + S2LENGTH)
 == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 /* Concatenate s1buf with s2buf, placing the result into */
 /* s1buf. Then compare s1buf with the expected result */
 /* in test1. */
 wcscat(s1buf, s2buf);

 for (i = 0; i < S1LENGTH + S2LENGTH - 2; i++) {
 /* Check that each character is correct */
 if (test1[i] != s1buf[i]) {
 printf("Error in wcscat\n");
 exit(EXIT_FAILURE);
 }
 }

 printf("Concatenated string: <%S>\n", s1buf);
}
```

Running the example produces the following result:

```
Concatenated string: <abcmnexyz orthis>
```

---

## wcschr

Scans for a wide character in a specified wide-character string.

### Format

```
#include <wchar.h>

wchar_t *wcschr (const wchar_t *wstr, wchar_t wc);
```

### Function Variants

The `wcschr` function has variants named `_wcschr32` and `_wcschr64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**wstr**  
A pointer to a null-terminated wide-character string.

**wc**  
A character of type `wchar_t`.

### Description

The `wcschr` function returns the address of the first occurrence of a specified wide character in a null-terminated wide-character string. The terminating null character is considered to be part of the string. See also `wcsrchr` in this section.

### Return Values

|                   |                                                                      |
|-------------------|----------------------------------------------------------------------|
| <code>x</code>    | The address of the first occurrence of the specified wide character. |
| <code>NULL</code> | Indicates that the wide character does not occur in the string.      |

### Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

#define BUFF_SIZE 50

main()
{
 int i;
 wchar_t s1buf[BUFF_SIZE];
 wchar_t *status;

 /* Initialize the buffer */
 if (mbstowcs(s1buf, "abcdefghijkl lkjihgfedcba", BUFF_SIZE)
 == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }
}
```

## wcschr

```
/* This program checks the wcschr function by incrementally */
/* going through a string that ascends to the middle and */
/* then descends towards the end. */
for (i = 0; (slbuf[i] != '\0') && (slbuf[i] != ' '); i++) {
 status = wcschr(slbuf, slbuf[i]);
 /* Check for pointer to leftmost character - test 1. */
 if (status != &slbuf[i]) {
 printf("Error in wcschr\n");
 exit(EXIT_FAILURE);
 }
}
printf("Program completed successfully\n");
}
```

When this example program is run, it produces the following result:

```
Program completed successfully
```

---

## wcscmp

Compares two wide-character strings. It returns an integer that indicates if the strings are different, and how they differ.

### Format

```
#include <wchar.h>

int wcscmp (const wchar_t *wstr_1, const wchar_t *wstr_2);
```

### Arguments

**wstr\_1, wstr\_2**

Pointers to null-terminated wide-character strings.

### Description

The `wcscmp` function compares the wide characters in *wstr\_1* with those in *wstr\_2*. If the characters differ, the function returns:

- An integer less than 0, if the codepoint of the first differing character in *wstr\_1* is less than the codepoint of the corresponding character in *wstr\_2*
- An integer greater than 0, if the codepoint of the first differing character in *wstr\_1* is greater than the codepoint of the corresponding character in *wstr\_2*

If the wide-characters strings are identical, the function returns zero.

Unlike the `wscoll` function, the `wcscmp` function compares the string based on the binary value of each wide character.

See also `wcsncmp` in this section.

### Return Values

|     |                                                              |
|-----|--------------------------------------------------------------|
| < 0 | Indicates that <i>wstr_1</i> is less than <i>wstr_2</i> .    |
| = 0 | Indicates that <i>wstr_1</i> equals <i>wstr_2</i> .          |
| > 0 | Indicates that <i>wstr_1</i> is greater than <i>wstr_2</i> . |

---

**wcscoll**

Compares two wide-character strings and returns an integer that indicates if the strings differ, and how they differ. The function uses the collating information in the LC\_COLLATE category of the current locale to determine how the comparison is performed.

**Format**

```
#include <wchar.h>

int wcscoll (const wchar_t *ws1, const wchar_t *ws2);
```

**Arguments**

**ws1, ws2**  
Pointers to wide-character strings.

**Description**

The `wcscoll` function, unlike `wscmp`, compares two strings in a locale-dependent manner. Because no value is reserved for error indication, the application must check for one by setting `errno` to 0 before the function call and testing it after the call.

See also `wcsxfrm` in this section.

**Return Values**

|     |                                                        |
|-----|--------------------------------------------------------|
| < 0 | Indicates that <i>ws1</i> is less than <i>ws2</i> .    |
| 0   | Indicates that the strings are equal.                  |
| > 0 | Indicates that <i>ws1</i> is greater than <i>ws2</i> . |

---

## wcscpy

Copies the wide-character string *source*, including the terminating null character, into *dest*.

### Format

```
#include <wchar.h>

wchar_t *wcscpy (wchar_t *dest, const wchar_t *source);
```

### Function Variants

The `wcscpy` function has variants named `_wcscpy32` and `_wcscpy64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**

Pointer to the null-terminated wide-character destination string.

**source**

Pointer to the null-terminated wide-character source string.

### Description

The `wcscpy` function copies *source* into *dest*, and stops after copying *source*'s null character. If copying takes place between two overlapping strings, the behavior is undefined.

See also `wcsncpy` in this section.

### Return Value

|   |                                |
|---|--------------------------------|
| x | The address of <i>source</i> . |
|---|--------------------------------|

---

**wcscspn**

Compares the characters in a wide-character string against a set of wide characters. The function returns the length of the initial substring that is comprised entirely of characters that are not in the set of wide characters.

**Format**

```
#include <wchar.h>

size_t wcscspn (const wchar_t *wstr1, const wchar_t *wstr2);
```

**Arguments****wstr1**

A pointer to a null-terminated wide-character string. If this is a null string, 0 is returned.

**wstr2**

A pointer to a null-terminated wide-character string that contains the set of wide characters for which the function will search.

**Description**

The `wcscspn` function scans the wide characters in the string pointed to by *wstr1* until it encounters a character found in *wstr2*. The function returns the length of the initial segment of *wstr1* that is formed by characters not found in *wstr2*.

**Return Value**

|   |                            |
|---|----------------------------|
| x | The length of the segment. |
|---|----------------------------|

**Example**

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

/* This test sets up 2 strings, buffer and w_string, and */
/* then uses wcscspn() to calculate the maximum segment */
/* of w_string, which consists entirely of characters */
/* NOT from buffer. */

#define BUFF_SIZE 20
#define STRING_SIZE 50

main()
{
 wchar_t buffer[BUFF_SIZE];
 wchar_t w_string[STRING_SIZE];
 size_t result;

 /* Initialize the buffer */

 if (mbstowcs(buffer, "abcdefg", BUFF_SIZE) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 /* Initialize the string */
```



```
 if (mbstowcs(w_string, "jklmabcjklabcdehijklmno", STRING_SIZE)
 == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }
 /* Using wcscspn - work out the largest string in w_string */
 /* which consists entirely of characters NOT from buffer */
 result = wcscspn(w_string, buffer);
 printf("Longest segment NOT found in w_string is: %d", result);
}
```

Running the example program produces the following result:

Longest segment NOT found in w\_string is: 4

---

## wcsftime

Uses date and time information stored in a `tm` structure to create a wide-character output string. The format of the output string is controlled by a format string.

### Format

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

```
size_t wcsftime (wchar_t *wcs, size_t maxsize, const char *format, const struct tm *timeptr); (XPG4)
```

```
size_t wcsftime (wchar_t *wcs, size_t maxsize, const wchar_t *format, const struct tm *timeptr); (ISO C)
```

### Function Variants

Compiling with the `_DECC_V4_SOURCE` and `_VMS_V6_SOURCE` feature-test macros defined enables a local-time-based entry point to the `wcsftime` function that is equivalent to the behavior before OpenVMS Version 7.0.

### Arguments

#### **wcs**

A pointer to the resultant wide-character string.

#### **maxsize**

The maximum number of wide characters to be stored in the resultant string.

#### **format**

A pointer to the string that controls the format of the output string. For the XPG4 interface, this argument is a pointer to a constant character string. For the ISO C interface, it is a pointer to a constant wide-character string.

#### **timeptr**

A pointer to the local time structure. The `tm` structure is defined in the `<time.h>` header file.

### Description

The `wcsftime` function uses data in the structure pointed to by `timeptr` to create the wide-character string pointed to by `wcs`. A maximum of `maxsize` wide characters is copied to `wcs`.

The format string consists of zero or more conversion specifications and ordinary characters. All ordinary characters (including the terminating null character) are copied unchanged into the output string. A conversion specification defines how data in the `tm` structure is formatted in the output string.

A conversion specification consists of a percent (%) character followed by one or more optional characters (see Table REF-13), and ending with a conversion specifier (see Table REF-14). If any of the optional characters listed in Table REF-13 are specified, they must appear in the order shown in the table.

**Table REF–13 Optional Elements of wcsftime Conversion Specifications**

| Element     | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| –           | Optional with the field width to specify that the field is left-justified and padded with spaces. This cannot be used with the 0 element.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 0           | Optional with the field width to specify that the field is right-justified and padded with zeros. This cannot be used with the – element.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| field width | A decimal integer that specifies the maximum field width                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| .precision  | <p>A decimal integer that specifies the precision of data in a field.</p> <p>For the d, H, I, j, m, M, o, S, U, w, W, y, and Y conversion specifiers, the precision specifier is the minimum number of digits to appear in the field. If the conversion specification has fewer digits than that specified by the precision, leading zeros are added.</p> <p>For the a, A, b, B, c, D, E, h, n, N, p, r, t, T, x, X, Z, and % conversion specifiers, the precision specifier is the maximum number of wide characters to appear in the field. If the conversion specification has more characters than that specified by the the precision, characters are truncated on the right.</p> <p>The default precision for the d, H, I, m, M, o, S, U, w, W, y, and Y conversion specifiers is 2, and the default precision for the j conversion specifier is 3.</p> |

Note that the list of optional elements of conversion specifications from Table REF–13 are Compaq extensions to the XPG4 specification.

Table REF–14 lists the conversion specifiers. The `wcsftime` function uses fields in the `LC_TIME` category of the program's current locale to provide a value. For example, if `%B` is specified, the function accesses the *mon* field in `LC_TIME` to find the full month name for the month specified in the `tm` structure. The result of using invalid conversion specifiers is undefined.

**Table REF–14 wcsftime Conversion Specifiers**

| Specifier | Replaced by                                                                                                 |
|-----------|-------------------------------------------------------------------------------------------------------------|
| a         | The locale's abbreviated weekday name.                                                                      |
| A         | The locale's full weekday name.                                                                             |
| b         | The locale's abbreviated month name.                                                                        |
| B         | The locale's full month name.                                                                               |
| c         | The locale's appropriate date and time representation.                                                      |
| C         | The century number (the year divided by 100 and truncated to an integer) as a decimal number (00 – 99).     |
| d         | The day of the month as a decimal number (01 – 31).                                                         |
| D         | Same as <code>%m/%d/%y</code> .                                                                             |
| e         | The day of the month as a decimal number (1 – 31) in a 2-digit field with the leading space character fill. |
| Ec        | The locale's alternative date and time representation.                                                      |

(continued on next page)

**Table REF–14 (Cont.) wcsftime Conversion Specifiers**

| <b>Specifier</b> | <b>Replaced by</b>                                                                                                                                                                                                                                                                                                                           |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EC               | The name of the base year (period) in the locale's alternative representation.                                                                                                                                                                                                                                                               |
| Ex               | The locale's alternative date representation.                                                                                                                                                                                                                                                                                                |
| Ey               | The offset from the base year (%EC) in the locale's alternative representation.                                                                                                                                                                                                                                                              |
| EY               | The locale's full alternative year representation.                                                                                                                                                                                                                                                                                           |
| h                | Same as %b.                                                                                                                                                                                                                                                                                                                                  |
| H                | The hour (24-hour clock) as a decimal number (00 – 23).                                                                                                                                                                                                                                                                                      |
| I                | The hour (12-hour clock) as a decimal number (01 – 12).                                                                                                                                                                                                                                                                                      |
| j                | The day of the year as a decimal number (001 – 366).                                                                                                                                                                                                                                                                                         |
| m                | The month as a decimal number (01 – 12).                                                                                                                                                                                                                                                                                                     |
| M                | The minute as a decimal number (00 – 59).                                                                                                                                                                                                                                                                                                    |
| n                | The new-line character.                                                                                                                                                                                                                                                                                                                      |
| Od               | The day of the month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                         |
| Oe               | The date of the month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                        |
| OH               | The hour (24-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                     |
| OI               | The hour (12-hour clock) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                     |
| Om               | The month using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                    |
| OM               | The minutes using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                  |
| OS               | The seconds using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                                  |
| Ou               | The weekday as a number in the locale's alternative representation (Monday=1).                                                                                                                                                                                                                                                               |
| OU               | The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.                                                                                                                                                                                                                            |
| OV               | The week number of the year (Monday as the first day of the week) as a decimal number (01 –53) using the locale's alternative numeric symbols. If the week containing January 1 has four or more days in the new year, it is considered as week 1. Otherwise, it is considered as week 53 of the previous year, and the next week is week 1. |
| Ow               | The weekday as a number (Sunday=0) using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                           |
| OW               | The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.                                                                                                                                                                                                                            |
| Oy               | The year without the century using the locale's alternative numeric symbols.                                                                                                                                                                                                                                                                 |

(continued on next page)

Table REF–14 (Cont.) wcsftime Conversion Specifiers

| Specifier | Replaced by                                                                                                                                                                                                                                                                                    |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| p         | The locale's equivalent of the AM/PM designations associated with a 12-hour clock.                                                                                                                                                                                                             |
| r         | The time in AM/PM notation.                                                                                                                                                                                                                                                                    |
| R         | The time in 24-hour notation (%H:%M).                                                                                                                                                                                                                                                          |
| S         | The second as a decimal number (00 – 61).                                                                                                                                                                                                                                                      |
| t         | The tab character.                                                                                                                                                                                                                                                                             |
| T         | The time (%H:%M:%S).                                                                                                                                                                                                                                                                           |
| u         | The weekday as a decimal number between 1 and 7 (Monday=1).                                                                                                                                                                                                                                    |
| U         | The week number of the year (the first Sunday as the first day of week 1) as a decimal number (00 – 53).                                                                                                                                                                                       |
| V         | The week number of the year (Monday as the first day of the week) as a decimal number (00 – 53). If the week containing January 1 has four or more days in the new year, it is considered as week 1. Otherwise, it is considered as week 53 of the previous year, and the next week is week 1. |
| w         | The weekday as a decimal number (0 [Sunday] – 6).                                                                                                                                                                                                                                              |
| W         | The week number of the year (the first Monday as the first day of week 1) as a decimal number (00 – 53).                                                                                                                                                                                       |
| x         | The locale's appropriate date representation                                                                                                                                                                                                                                                   |
| X         | The locale's appropriate time representation                                                                                                                                                                                                                                                   |
| y         | The year without century as a decimal number (00 – 99).                                                                                                                                                                                                                                        |
| Y         | The year with century as a decimal number.                                                                                                                                                                                                                                                     |
| Z         | Time-zone name or abbreviation. If time-zone information is not available, no character is output.                                                                                                                                                                                             |
| %         | Literal % character.                                                                                                                                                                                                                                                                           |

## Return Values

|   |                                                                                                                              |
|---|------------------------------------------------------------------------------------------------------------------------------|
| x | The number of wide characters placed into the array pointed to by <i>wcs</i> , not including the terminating null character. |
| 0 | Indicates an error occurred. The contents of the array are indeterminate.                                                    |

## Example

```

/* Exercise the wcsftime formatting routine. */
/* NOTE: the format string is an "L" (or wide character) */
/* string indicating that this call is NOT in */
/* the XPG4 format, but rather in ISO C format. */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <wchar.h>
#include <locale.h>
#include <errno.h>

```

## wcsftime

```
#define NUM_OF_DATES 7
#define BUF_SIZE 256

/* This program formats a number of different dates, once using the */
/* C locale and then using the fr_FR.ISO8859-1 locale. Date and time */
/* formatting is done using wcsftime(). */

main()
{
 int count,
 i;
 wchar_t buffer[BUF_SIZE];
 struct tm *tm_ptr;
 time_t time_list[NUM_OF_DATES] =
 {500, 68200000, 694223999,
 694224000, 704900000, 705000000,
 705900000};

 /* Display dates using the C locale */
 printf("\nUsing the C locale:\n\n");
 setlocale(LC_ALL, "C");

 for (i = 0; i < NUM_OF_DATES; i++) {
 /* Convert to a tm structure */
 tm_ptr = localtime(&time_list[i]);

 /* Format the date and time */
 count = wcsftime(buffer, BUF_SIZE, L>Date: %A %d %B %Y\nTime: %T%n\n",
 tm_ptr);
 if (count == 0) {
 perror("wcsftime");
 exit(EXIT_FAILURE);
 }

 /* Print the result */
 printf("%S", buffer);
 }

 /* Display dates using the fr_FR.ISO8859-1 locale */
 printf("\nUsing the fr_FR.ISO8859-1 locale:\n\n");
 setlocale(LC_ALL, "fr_FR.ISO8859-1");

 for (i = 0; i < NUM_OF_DATES; i++) {
 /* Convert to a tm structure */
 tm_ptr = localtime(&time_list[i]);

 /* Format the date and time */
 count = wcsftime(buffer, BUF_SIZE, L>Date: %A %d %B %Y\nTime: %T%n\n",
 tm_ptr);
 if (count == 0) {
 perror("wcsftime");
 exit(EXIT_FAILURE);
 }

 /* Print the result */
 printf("%S", buffer);
 }
}
```

Running the example program produces the following result:

Using the C locale:

Date: Thursday 01 January 1970  
Time: 00:08:20

Date: Tuesday 29 February 1972  
Time: 08:26:40

Date: Tuesday 31 December 1991  
Time: 23:59:59

Date: Wednesday 01 January 1992  
Time: 00:00:00

Date: Sunday 03 May 1992  
Time: 13:33:20

Date: Monday 04 May 1992  
Time: 17:20:00

Date: Friday 15 May 1992  
Time: 03:20:00

Using the fr\_FR.ISO8859-1 locale:

Date: jeudi 01 janvier 1970  
Time: 00:08:20

Date: mardi 29 février 1972  
Time: 08:26:40

Date: mardi 31 décembre 1991  
Time: 23:59:59

Date: mercredi 01 janvier 1992  
Time: 00:00:00

Date: dimanche 03 mai 1992  
Time: 13:33:20

Date: lundi 04 mai 1992  
Time: 17:20:00

Date: vendredi 15 mai 1992  
Time: 03:20:00

## wcslen

---

## wcslen

Returns the number of wide characters in a wide-character string. The returned length does not include the terminating null character.

### Format

```
#include <wchar.h>

size_t wcslen (const wchar_t *wstr);
```

### Argument

**wstr**

A pointer to a null-terminated wide-character string.

### Return Value

|   |                                                                                         |
|---|-----------------------------------------------------------------------------------------|
| x | The length of the wide-character string, excluding the terminating null wide character. |
|---|-----------------------------------------------------------------------------------------|



---

## wcsncat

Concatenates a counted number of wide-characters from one string to another.

### Format

```
#include <wchar.h>

wchar_t *wcsncat (wchar_t *wstr_1, const wchar_t *wstr_2, size_t maxchar);
```

### Function Variants

The `wcsncat` function has variants named `_wcsncat32` and `_wcsncat64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**wstr\_1, wstr\_2**

Pointers to null-terminated wide-character strings.

**maxchar**

The maximum number of wide characters from *wstr\_2* that are copied to *wstr\_1*. If *maxchar* is 0, no characters are copied from *wstr\_2*.

### Description

The `wcsncat` function appends wide characters from the wide-character string *wstr\_2* to the end of *wstr\_1*, up to a maximum of *maxchar* characters. A terminating null wide character is always appended to the result of the `wcsncat` function. Therefore, the maximum number of wide characters that can end up in *wstr\_1* is `wcslen(wstr_1) + maxchar + 1`.

See also `wcscat` in this section.

### Return Value

|   |                                                                                                          |
|---|----------------------------------------------------------------------------------------------------------|
| x | The first argument, <i>wstr_1</i> , which is assumed to be large enough to hold the concatenated result. |
|---|----------------------------------------------------------------------------------------------------------|

### Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

/* This program concatenates two wide-character strings using */
/* the wcsncat function, and then manually compares the result */
/* to the expected result */

#define S1LENGTH 10
#define S2LENGTH 8
#define SIZE 3
```

## wcsncat

```
main()
{
 int i;
 wchar_t s1buf[S1LENGTH + S2LENGTH];
 wchar_t s2buf[S2LENGTH];
 wchar_t test1[S1LENGTH + S2LENGTH];

 /* Initialize the three wide-character strings */

 if (mbstowcs(s1buf, "abcmnxyz", S1LENGTH) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 if (mbstowcs(s2buf, " orthis", S2LENGTH) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 if (mbstowcs(test1, "abcmnxyz orthis", S1LENGTH + SIZE)
 == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 /* Concatenate s1buf with SIZE characters from s2buf, placing the */
 /* result into s1buf. Then compare s1buf with the expected result */
 /* in test1. */
 wcsncat(s1buf, s2buf, SIZE);

 for (i = 0; i <= S1LENGTH + SIZE - 2; i++) {
 /* Check that each character is correct */
 if (test1[i] != s1buf[i]) {
 printf("Error in wcsncat\n");
 exit(EXIT_FAILURE);
 }
 }

 printf("Concatenated string: <%S>\n", s1buf);
}
```

Running the example produces the following result:

```
Concatenated string: <abcmnxyz or>
```

---

## wcsncmp

Compares not more than *maxchar* characters of two wide-character strings. It returns an integer that indicates if the strings are different, and how they differ.

### Format

```
#include <wchar.h>

int wcsncmp (const wchar_t *wstr_1, const wchar_t *wstr_2, size_t maxchar);
```

### Arguments

**wstr\_1, wstr\_2**

Pointers to null-terminated wide-character strings.

**maxchar**

The maximum number of characters to search in both *wstr\_1* and *wstr\_2*. If *maxchar* is 0, no comparison is performed and 0 is returned (the strings are considered equal).

### Description

The strings are compared until a null character is encountered, the strings differ, or *maxchar* is reached. If characters differ, *wcsncmp* returns:

- An integer less than 0 if the codepoint of the first differing character in *wstr\_1* is less than the codepoint of the corresponding character in *wstr\_2*
- An integer greater than 0 if the codepoint of the first differing character in *wstr\_1* is greater than the codepoint of the corresponding character in *wstr\_2*

If no differences are found after comparing *maxchar* characters, the function returns zero.

See also *wcscmp* in this section.

### Return Values

|     |                                                              |
|-----|--------------------------------------------------------------|
| < 0 | Indicates that <i>wstr_1</i> is less than <i>wstr_2</i> .    |
| 0   | Indicates that <i>wstr_1</i> equals <i>wstr_2</i> .          |
| > 0 | Indicates that <i>wstr_1</i> is greater than <i>wstr_2</i> . |

---

## wcsncpy

Copies wide characters from *source* into *dest*. The function copies up to a maximum of *maxchar* characters.

### Format

```
#include <wchar.h>

wchar_t *wcsncpy (wchar_t *dest, const wchar_t *source, size_t maxchar);
```

### Function Variants

The `wcsncpy` function has variants named `_wcsncpy32` and `_wcsncpy64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**

Pointer to the null-terminated wide-character destination string.

**source**

Pointer to the null-terminated wide-character source string.

**maxchar**

The maximum number of wide characters to copy from *source* to *dest*.

### Description

The `wcsncpy` function copies no more than *maxchar* characters from *source* to *dest*. If *source* contains less than *maxchar* characters, null characters are added to *dest* until *maxchar* characters have been written to *dest*.

If *source* contains *maxchar* or more characters, as many characters as possible are copied to *dest*. The null terminator of *source* is not copied to *dest*.

See also `wscpy` in this section.

### Return Value

|   |                              |
|---|------------------------------|
| x | The address of <i>dest</i> . |
|---|------------------------------|

---

## wcpbrk

Searches a wide-character string for the first occurrence of one of a specified set of wide characters.

### Format

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

```
wchar_t *wcpbrk (const wchar_t *wstr, const wchar_t *charset);
```

### Function Variants

The `wcpbrk` function has variants named `_wcpbrk32` and `_wcpbrk64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**wstr**

A pointer to a wide-character string. If this is a null string, NULL is returned.

**charset**

A pointer to a wide-character string containing the set of wide characters for which the function will search.

### Description

The `wcpbrk` function scans the wide characters in the string, stops when it encounters a wide character found in *charset*, and returns the address of the first character in the string that appears in the character set.

### Return Values

|      |                                                                           |
|------|---------------------------------------------------------------------------|
| x    | The address of the first wide character in the string that is in the set. |
| NULL | Indicates that none of the characters are in <i>charset</i> .             |

---

**wcsrchr**

Scans for the last occurrence of a wide character in a given string.

**Format**

```
#include <wchar.h>

wchar_t *wcsrchr (const wchar_t *wstr, wchar_t wc);
```

**Function Variants**

The `wcsrchr` function has variants named `_wcsrchr32` and `_wcsrchr64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

**Arguments****wstr**

A pointer to a null-terminated wide-character string.

**wc**

A character of type `wchar_t`.

**Description**

The `wcsrchr` function returns the address of the last occurrence of a given wide character in a null-terminated wide-character string. The terminating null character is considered to be part of the string.

See also `wcschr` in this section.

**Return Values**

|                   |                                                                     |
|-------------------|---------------------------------------------------------------------|
| <code>x</code>    | The address of the last occurrence of the specified wide character. |
| <code>NULL</code> | Indicates that the wide character does not occur in the string.     |

**Example**

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

#define BUFF_SIZE 50
#define STRING_SIZE 6

main()
{
 int i;
 wchar_t s1buf[BUFF_SIZE],
 w_string[STRING_SIZE];
 wchar_t *status;
 wchar_t *pbuf = s1buf;

 /* Initialize the buffer */
 if (mbstowcs(s1buf, "hijklabcdefg ytuhiijklfedcba", BUFF_SIZE)
```

```

 == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 /* Initialize the string to be searched for */

 if (mbstowcs(w_string, "hijkl", STRING_SIZE) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 /* This program checks the wcsrchr function by searching for */
 /* the last occurrence of a string in the buffer slbuf and */
 /* prints out the contents of slbuff from the location of */
 /* the string found. */
 status = wcsrchr(slbuf, w_string[0]);
 /* Check for pointer to start of rightmost character string. */
 if (status == pbuf) {
 printf("Error in wcsrchr\n");
 exit(EXIT_FAILURE);
 }

 printf("Program completed successfully\n");
 printf("String found : [%S]\n", status);
 }

```

Running the example produces the following result:

```

Program completed successfully
String found : [hijklfedcba]

```

---

**wcsrtombs**

Converts a sequence of wide characters into a sequence of corresponding multibyte characters.

**Format**

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

```
size_t wcsrtombs (char *dst, const wchar_t **src, size_t len, mbstate_t *ps);
```

**Function Variants**

The `wcsrtombs` function has variants named `_wcsrtombs32` and `_wcsrtombs64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

**Arguments****dst**

A pointer to the destination array for converted multibyte character sequence.

**src**

An address of the pointer to an array containing the sequence of wide characters to be converted.

**len**

The maximum number of bytes that can be stored in the array pointed to by *dst*.

**ps**

A pointer to the `mbstate_t` object. If a NULL pointer is specified, the function uses its internal `mbstate_t` object. `mbstate_t` is an opaque datatype intended to keep the conversion state for the state-dependent codesets.

**Description**

The `wcsrtombs` function converts a sequence of wide characters from the array indirectly pointed to by *src* into a sequence of corresponding multibyte characters, beginning in the conversion state described by the object pointed to by *ps*.

If *dst* is not a NULL pointer, the converted characters are then stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases:

- When a code is reached that does not correspond to a valid multibyte character
- If *dst* is not a NULL pointer, when the next multibyte character would exceed the limit of *len* total bytes to be stored into the array pointed to by *dst*

Each conversion takes place as if by a call to the `wcrtomb` function.

If *dst* is not a NULL pointer, the pointer object pointed to by *src* is assigned either a NULL pointer, (if the conversion stopped because it reached a terminating null wide character) or the address just beyond the last wide character converted (if any). If conversion stopped because it reached a terminating null wide character, the resulting state described is the initial conversion state.



If the `wcsrtombs` function is called as a counting function, which means that *dst* is a `NULL` pointer, the value of the internal `mbstate_t` object will remain unchanged.

See also `wcrtomb` in this section.

## Return Values

|                 |                                                                                                                                                                                                        |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>  | The number of bytes stored in the resulting array, not including the terminating null (if any).                                                                                                        |
| <code>-1</code> | Indicates an encoding error—a character that does not correspond to a valid multibyte character was encountered; <code>errno</code> is set to <code>EILSEQ</code> ; the conversion state is undefined. |

---

**wcssp**

Compares the characters in a wide-character string against a set of wide characters. The function returns the length of the first substring comprised entirely of characters in the set of wide characters.

**Format**

```
#include <wchar.h>

size_t wcssp (const wchar_t *wstr1, const wchar_t *wstr2);
```

**Arguments****wstr1**

A pointer to a null-terminated wide-character string. If this string is a null string, 0 is returned.

**wstr2**

A pointer to a null-terminated wide-character string that contains the set of wide characters for which the function will search.

**Description**

The `wcssp` function scans the wide characters in the wide-character string pointed to by `wstr1` until it encounters a character not found in `wstr2`. The function returns the length of the first segment of `wstr1` formed by characters found in `wstr2`.

**Return Value**

|   |                            |
|---|----------------------------|
| x | The length of the segment. |
|---|----------------------------|

**Example**

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <string.h>

/* This test sets up 2 strings, buffer and w_string. It */
/* then uses wcssp() to calculate the maximum segment */
/* of w_string that consists entirely of characters */
/* from buffer. */

#define BUFF_SIZE 20
#define STRING_SIZE 50

main()
{
 wchar_t buffer[BUFF_SIZE];
 wchar_t w_string[STRING_SIZE];
 size_t result;

 /* Initialize the buffer */

 if (mbstowcs(buffer, "abcdefg", BUFF_SIZE) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }
}
```

```
/* Initialize the string */
if (mbstowcs(w_string, "abcdjklmabcjklabcdehijkl", STRING_SIZE)
 == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
}

/* Using wcssp - work out the largest string in w_string */
/* that consists entirely of characters from buffer */
result = wcssp(w_string, buffer);
printf("Longest segment found in w_string is: %d", result);
}
```

Running the example program produces the following result:

Longest segment found in w\_string is: 5

## wcsstr

---

## wcsstr

Locates the first occurrence in the string pointed to by *s1* of the sequence of wide characters in the string pointed to by *s2*.

### Format

```
#include <wchar.h>

wchar_t *wcsstr (const wchar_t *s1, const wchar_t *s2);
```

### Function Variants

The `wcsstr` function has variants named `_wcsstr32` and `_wcsstr64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**s1, s2**  
Pointers to null-terminated, wide-character strings.

### Description

If *s2* points to a wide-character string of zero length, the `wcsstr` function returns *s1*.

### Return Values

|      |                                               |
|------|-----------------------------------------------|
| x    | A pointer to the located string.              |
| NULL | Indicates an error; the string was not found. |

---

## wcstod

Converts a given wide-character string to a double-precision number.

### Format

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

```
double wcstod (const wchar_t *nptr, wchar_t **endptr);
```

### Arguments

#### **nptr**

A pointer to the wide-character string to be converted to a double-precision number.

#### **endptr**

The address of an object where the function can store the address of the first unrecognized wide character that terminates the scan. If *endptr* is a NULL pointer, the address of the first unrecognized wide character is not retained.

### Description

The `wcstod` function recognizes an optional sequence of white-space characters (as defined by `iswspace`), then an optional plus or minus sign, then a sequence of digits optionally containing a radix character, then an optional letter (`e` or `E`) followed by an optionally signed integer. The first unrecognized character ends the conversion.

The string is interpreted by the same rules used to interpret floating constants.

The radix character is defined in the program's current locale (category `LC_NUMERIC`).

This function returns the converted value. For `wcstod`, overflows are accounted for in the following manner:

- If the correct value causes an overflow, `HUGE_VAL` (with a plus or minus sign according to the sign of the value) is returned and `errno` is set to `ERANGE`.
- If the correct value causes an underflow, 0 is returned and `errno` is set to `ERANGE`.

If the string starts with an unrecognized wide character, *endptr* is set to *nptr* and a 0 value is returned.

### Return Values

x

The converted string.

## **wcstod**

|                        |                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0                      | <p>Indicates the conversion could not be performed. The function sets <code>errno</code> to one of:</p> <ul style="list-style-type: none"><li>• <code>EINVAL</code> – No conversion could be performed.</li><li>• <code>ERANGE</code> – The value would cause an underflow.</li><li>• <code>ENOMEM</code> – Not enough memory available for internal conversion buffer.</li></ul> |
| <code>±HUGE_VAL</code> | <p>Overflow occurred; <code>errno</code> is set to <code>ERANGE</code>.</p>                                                                                                                                                                                                                                                                                                       |

---

## wcstok

Locates text tokens in a given wide-character string.

### Format

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

```
wchar_t *wcstok (wchar_t *ws1, const wchar_t *ws2); (XPG4)
```

```
wchar_t *wcstok (wchar_t *ws1, const wchar_t *ws2, wchar_t **ptr); (ISO C)
```

### Function Variants

The `wcstok` function has variants named `_wcstok32` and `_wcstok64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **ws1**

A pointer to a wide-character string containing 0 or more text tokens.

#### **ws2**

A pointer to a separator string consisting of one or more wide characters. The separator string can differ from call to call.

#### **ptr**

ISO C Standard only. Used only when *ws1* is NULL, *ptr* is a caller-provided `wchar_t` pointer into which `wcstok` stores information necessary for it to continue scanning the same wide-character string.

### Description

A sequence of calls to `wcstok` breaks the wide-character string pointed to by *ws1* into a sequence of tokens, each of which is delimited by a wide character from the wide-character string pointed to by *ws2*.

The `wcstok` function keeps track of its position in the wide-character string between calls and, as successive calls are made, the function works through the wide-character string, identifying the text token following the one identified by the previous call.

Tokens in *ws1* are delimited by null characters that `wcstok` inserts into *ws1*. Therefore, *ws1* cannot be a const object.

The following sections describe differences between the XPG4 Standard and ISO C Standard interface to `wcstok`.

#### **XPG4 Standard Behavior**

The first call to the `wcstok` function searches the wide-character string for the first character that is *not* found in the separator string pointed to by *ws2*. The first call returns a pointer to the first wide character in the first token and writes a null wide character into *ws1* immediately following the returned token.

Subsequent calls to `wcstok` search for a wide character that *is* in the separator string pointed to by `ws2`. Each subsequent call (with the value of the first argument remaining NULL) returns a pointer to the next token in the string originally pointed to by `ws1`. When no tokens remain in the string, `wcstok` returns a NULL pointer.

### ISO C Standard Behavior

For the first call in the sequence, `ws1` points to a wide-character string. In subsequent calls for the same string, `ws1` is NULL. When `ws1` is NULL, the value pointed to by `ptr` matches that stored by the previous call for the same wide-character string. Otherwise, the value pointed to by `ptr` is ignored.

The first call in the sequence searches the wide-character string pointed to by `ws1` for the first wide character that is *not* contained in the current separator wide-character string pointed to by `ws2`. If no such wide character is found, then there are no tokens in the wide-character string pointed to by `ws1`, and `wcstok` returns a NULL pointer.

The `wcstok` function then searches from there for a wide character that *is* contained in the current separator wide-character string. If no such wide character is found, the current token extends to the end of the wide-character string pointed to by `ws1`, and subsequent searches in the same wide-character string for a token return a NULL pointer. If such a wide character is found, it is overwritten by a null wide character, which terminates the current token.

In all cases, `wcstok` stores sufficient information in the pointer pointed to by `ptr` so that subsequent calls with a NULL pointer for `ws1` and the unmodified pointer value for `ptr` start searching just past the element overwritten by a null wide character (if any).

## Return Values

|                |                                              |
|----------------|----------------------------------------------|
| <code>x</code> | A pointer to the first character of a token. |
| NULL           | Indicates that no token was found.           |

## Examples

```
1. /* XPG4 version of wcstok call */
 #include <wchar.h>
 #include <string.h>
 #include <stdio.h>

 main()
 {
 wchar_t str[] = L"...ab..cd,,ef.hi";

 printf("%S\n", wcstok(str, L"."));
 printf("%S\n", wcstok(NULL, L","));
 printf("%S\n", wcstok(NULL, L",."));
 printf("%S\n", wcstok(NULL, L",."));
 }
```



```

2. /* ISO C version of wcstok call */
 #include <wchar.h>
 #include <string.h>
 #include <stdio.h>

 main()
 {
 wchar_t str[] = L"...ab..cd,,ef.hi";
 wchar_t *savptr = NULL;

 printf("%S\n", wcstok(str, L".", &savptr));
 printf("%S\n", wcstok(NULL, L",", &savptr));
 printf("%S\n", wcstok(NULL, L".", &savptr));
 printf("%S\n", wcstok(NULL, L".", &savptr));
 }

```

Running this example produces the following results:

```

$ $ RUN WCSTOK_EXAMPLE
|ab|
|.cd|
|ef|
|hi|
$

```

---

## wcstol

Converts a wide-character string in a specified base to a long integer value.

### Format

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

```
long int wcstol (const wchar_t *nptr, wchar_t **endptr, int base);
```

### Function Variants

The `wcstol` function has variants named `_wcstol32` and `_wcstol64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### **nptr**

A pointer to the wide-character string to be converted to a long integer.

#### **endptr**

The address of an object where the function can store a pointer to the first unrecognized character encountered in the conversion process (the character that follows the last character processed in the string being converted). If *endptr* is a NULL pointer, the address of the first unrecognized character is not retained.

#### **base**

The value, 2 through 36, to use as the base for the conversion.

If *base* is 16, leading zeros after the optional sign are ignored, and 0x or 0X is ignored.

If *base* is 0, the sequence of characters is interpreted by the same rules used to interpret an integer constant. After the optional sign:

- A leading 0 indicates octal conversion.
- A leading 0x or 0X indicates hexadecimal conversion.
- Any other combination of leading characters indicates decimal conversion.

### Description

The `wcstol` function recognizes strings in various formats, depending on the value of the base. This function ignores any leading white-space characters (as defined by the `iswspace` function) in the given string. It recognizes an optional plus or minus sign, then a sequence of digits or letters that can represent an integer constant according to the value of the base. The first unrecognized character ends the conversion.

## Return Values

|                      |                                                                                                                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x                    | The converted value.                                                                                                                                                                                                                                                  |
| 0                    | Indicates that the string starts with an unrecognized wide character or that the value for <i>base</i> is invalid. If the string starts with an unrecognized wide character, <i>*endptr</i> is set to <i>nptr</i> . The function sets <i>errno</i> to <i>EINVAL</i> . |
| LONG_MAX or LONG_MIN | Indicates that the converted value would cause a positive or negative overflow, respectively. The function sets <i>errno</i> to <i>ERANGE</i> .                                                                                                                       |

---

## wcstombs

Converts a sequence of wide-character codes to a sequence of multibyte characters.

### Format

```
#include <stdlib.h>
```

```
size_t wcstombs (char *s, const wchar_t *pwcs, size_t n);
```

### Arguments

**s**

A pointer to the array containing the resulting multibyte characters.

**pwcs**

A pointer to the array containing the sequence of wide-character codes.

**n**

The maximum number of bytes to be stored in the array pointed to by *s*.

### Description

The `wcstombs` function converts a sequence of codes corresponding to multibyte characters from the array pointed to by *pwcs* to a sequence of multibyte characters that are stored into the array pointed to by *s*, up to a maximum of *n* bytes. The value returned is equal to the number of characters converted or a `-1` if an error occurred.

This function is affected by the `LC_CTYPE` category of the program's current locale. See also `wctomb` in this section.

If *s* is `NULL`, this function call is a counting operation and *n* is ignored.

### Return Values

**x**

The number of bytes stored in *s*, not including the null terminating byte. If *s* is `NULL`, `wcstombs` returns the number of bytes required for the multibyte character array.

**(size\_t) -1**

Indicates an error occurred. The function sets `errno` to `EILSEQ` – invalid character sequence, or a wide-character code does not correspond to a valid character.

---

## wcstoul

Converts the initial portion of the wide-character string pointed to by *nptr* to an unsigned long integer.

### Format

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

```
unsigned long int wcstoul (const wchar_t *nptr, wchar_t **endptr, int base);
```

### Function Variants

The `wcstoul` function has variants named `_wcstoul32` and `_wcstoul64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

#### ***nptr***

A pointer to the wide-character string to be converted to an unsigned long.

#### ***endptr***

The address of an object where the function can store the address of the first unrecognized character encountered in the conversion process (The character that follows the last character in the string being converted). If *endptr* is a NULL pointer, the address of the first unrecognized character is not retained.

#### ***base***

The value, 2 through 36, to use as the base for the conversion.

If *base* is 16, leading zeros after the optional sign are ignored, and 0x or 0X is ignored.

If *base* is 0, the sequence of characters is interpreted by the same rules used to interpret an integer constant: after the optional sign, a leading 0 indicates octal conversion, a leading 0x or 0X indicates hexadecimal conversion, and any other combination of leading characters indicates decimal conversion.

### Description

The `wcstoul` function recognizes strings in various formats, depending on the value of the base. It ignores any leading white-space characters (as defined by the `iswspace` function) in the string. It recognizes an optional plus or minus sign, then a sequence of digits or letters that may represent an integer constant according to the value of the base. The first unrecognized wide character ends the conversion.

## wcstoul

### Return Values

|           |                                                                                                                                                                                                                                                                       |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x         | The converted value.                                                                                                                                                                                                                                                  |
| 0         | Indicates that the string starts with an unrecognized wide character or that the value for <i>base</i> is invalid. If the string starts with an unrecognized wide character, <i>*endptr</i> is set to <i>nptr</i> . The function sets <i>errno</i> to <i>EINVAL</i> . |
| ULONG_MAX | Indicates that the converted value would cause an overflow. The function sets <i>errno</i> to <i>ERANGE</i> .                                                                                                                                                         |

### Example

```
#include <stdlib.h>
#include <stdio.h>
#include <wchar.h>
#include <errno.h>
#include <limits.h>

/* This test calls wcstoul() to convert a string to an */
/* unsigned long integer. wcstoul outputs the resulting */
/* integer and any characters that could not be converted. */
#define MAX_STRING 128

main()
{
 int base = 10,
 errno;
 char *input_string = "1234.56";
 wchar_t string_array[MAX_STRING],
 *ptr;
 size_t size;
 unsigned long int val;
 printf("base = [%d]\n", base);
 printf("String to convert = %s\n", input_string);
 if ((size = mbstowcs(string_array, input_string, MAX_STRING)) ==
 (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }
 printf("wchar_t string is = [%S]\n", string_array);
 errno = 0;
 val = wcstoul(string_array, &ptr, base);
 if (errno == 0) {
 printf("returned unsigned long int from wcstoul = [%u]\n", val);
 printf("wide char terminating scan(ptr) = [%S]\n\n", ptr);
 }
 if (errno == ERANGE) {
 perror("error value is :");
 printf("ULONG_MAX = [%u]\n", ULONG_MAX);
 printf("wcstoul failed, val = [%d]\n\n", val);
 }
}
```

Running the example program produces the following result:

```
base = [10]
String to convert = 1234.56
wchar_t string is = [1234.56]
returned unsigned long int from wcstoul = [1234]
wide char terminating scan(ptr) = [.56]
```

---

**WCSWCS**

Locates the first occurrence in the string pointed to by *wstr1* of the sequence of wide characters in the string pointed to by *wstr2*.

**Format**

```
#include <wchar.h>

wchar_t *wcswcs (const wchar_t *wstr1, const wchar_t *wstr2);
```

**Function Variants**

The `wcswcs` function has variants named `_wcswcs32` and `_wcswcs64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

**Arguments**

**wstr1, wstr2**

Pointers to null-terminated wide-character strings.

**Return Values**

|         |                                                         |
|---------|---------------------------------------------------------|
| Pointer | A pointer to the located wide-character string.         |
| NULL    | Indicates that the wide-character string was not found. |

**Example**

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

/* This test uses wcswcs() to find the occurrence of each */
/* sub wide-character string, string1 and string2, within */
/* the main wide-character string, lookin. */
#define BUF_SIZE 50

main()
{
 static char lookin[] = "that this is a test was at the end";
 char string1[] = "this",
 string2[] = "the end";
 wchar_t buffer[BUF_SIZE],
 input_buffer[BUF_SIZE];

 /* Convert lookin to wide-character format. */
 /* Buffer and print it out. */

 if (mbstowcs(buffer, lookin, BUF_SIZE) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 printf("Buffer to look in: %S\n", buffer);

 /* Convert string1 to wide-character format and use */
 /* wcswcs() to locate it within buffer */
}
```



```

if (mbstowcs(input_buffer, string1, BUF_SIZE) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
}

printf("this: %S\n", wswcs(buffer, input_buffer));
/* Convert string2 to wide-character format and use */
/* wswcs() to locate it within buffer */

if (mbstowcs(input_buffer, string2, BUF_SIZE) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
}

printf("the end: %S\n", wswcs(buffer, input_buffer));
exit(1);
}

```

Running this example produces the following results:

```

Buffer to look in: that this is a test was at the end
this: this is a test was at the end
the end: the end

```

## wcswidth

---

## wcswidth

Determines the number of printing positions on a display device that are required for a wide-character string.

### Format

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

```
int wcswidth (const wchar_t *pwcs, size_t n);
```

### Arguments

**pwcs**

A pointer to a wide-character string.

**n**

The maximum number of characters in the string.

### Description

The `wcswidth` function returns the number of printing positions required to display the first *n* characters of the string pointed to by *pwcs*. If there are less than *n* wide characters in the string, the function returns the number of positions required for the whole string.

### Return Values

|    |                                                                                                                           |
|----|---------------------------------------------------------------------------------------------------------------------------|
| x  | The number of printing positions required.                                                                                |
| 0  | If <i>pwcs</i> is a null character.                                                                                       |
| −1 | Indicates that one (or more) of the wide characters in the string pointed to by <i>pwcs</i> is not a printable character. |

---

## wcsxfrm

Changes a wide-character string such that the changed string can be passed to the `wcscmp` function and produce the same result as passing the unchanged string to the `wscoll` function.

### Format

```
#include <wchar.h>

size_t wcsxfrm (wchar_t *ws1, const wchar_t *ws2, size_t maxchar);
```

### Arguments

**ws1, ws2**

Pointers to wide-character strings.

**maxchar**

The maximum number of wide-characters, including the null wide-character terminator, allowed to be stored in *s1*.

### Description

The `wcsxfrm` function transforms the string pointed to by *ws2* and stores the resulting string in the array pointed to by *ws1*. No more than *maxchar* wide characters, including the null wide terminator, are placed into the array pointed to by *ws1*.

If the value of *maxchar* is less than the required size to store the transformed string (including the terminating null), the contents of the array pointed to by *ws1* is indeterminate. In such a case, the function returns the size of the transformed string.

If *maxchar* is zero, then, *ws1* is allowed to be a NULL pointer, and the function returns the required size of the *ws1* array before making the transformation.

The wide-character string comparison functions, `wscoll` and `wcscmp`, can produce different results given the same two wide-character strings to compare. This is because `wcscmp` does a straightforward comparison of the code point values of the characters in the strings, whereas `wscoll` uses the locale information to do the comparison. Depending on the locale, the `wscoll` comparison can be a multi-pass operation, which is slower than `wcscmp`.

The `wcsxfrm` function transforms wide character strings in such a way that if you pass two transformed strings to the `wcscmp` function, the result is the same as passing the two original strings to the `wscoll` function. The `wcsxfrm` function is useful in applications that need to do a large number of comparisons on the same wide-character strings using `wscoll`. In this case, it may be more efficient (depending on the locale) to transform the strings once using `wcsxfrm` and then use the `wcscmp` function to do comparisons.

**Return Values**

|                           |                                                                                                                                                                                                  |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>            | Length of the resulting string pointed to by <i>ws1</i> , not including the terminating null character.                                                                                          |
| <code>(size_t) - 1</code> | Indicates an error occurred. The function sets <code>errno</code> to <code>EINVAL</code> – The string pointed to by <i>ws2</i> contains characters outside the domain of the collating sequence. |

**Example**

```
#include <wchar.h>
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

/* This program verifies that two transformed strings, */
/* when passed through wcsxfrm and then compared, provide */
/* the same result as if passed through wcscoll without */
/* any transformation. */

#define BUFF_SIZE 20

main()
{
 wchar_t w_string1[BUFF_SIZE];
 wchar_t w_string2[BUFF_SIZE];
 wchar_t w_string3[BUFF_SIZE];
 wchar_t w_string4[BUFF_SIZE];
 int errno;
 int coll_result;
 int wcscmp_result;
 size_t wcsxfrm_result1;
 size_t wcsxfrm_result2;

 /* setlocale to French locale */
 if (setlocale(LC_ALL, "fr_FR.ISO8859-1") == NULL) {
 perror("setlocale");
 exit(EXIT_FAILURE);
 }

 /* Convert each of the strings into wide-character format. */

 if (mbstowcs(w_string1, "<a'>bcd", BUFF_SIZE) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 if (mbstowcs(w_string2, "abcz", BUFF_SIZE) == (size_t)-1) {
 perror("mbstowcs");
 exit(EXIT_FAILURE);
 }

 /* Collate string 1 and string 2 and store the result. */
 errno = 0;
 coll_result = wcscoll(w_string1, w_string2);
 if (errno) {
 perror("wcscoll");
 exit(EXIT_FAILURE);
 }
 else {
```

```

/* Transform the strings (using wcsxfrm) into */
/* w_string3 and w_string4. */
wcsxfrm_result1 = wcsxfrm(w_string3, w_string1, BUFF_SIZE);
if (wcsxfrm_result1 == ((size_t) - 1))
 perror("wcsxfrm");
else if (wcsxfrm_result1 > BUFF_SIZE) {
 perror("\n** String is too long **\n");
 exit(EXIT_FAILURE);
}
else {
 wcsxfrm_result2 = wcsxfrm(w_string4, w_string2, BUFF_SIZE);
 if (wcsxfrm_result2 == ((size_t) - 1)) {
 perror("wcsxfrm");
 exit(EXIT_FAILURE);
 }
 else if (wcsxfrm_result2 > BUFF_SIZE) {
 perror("\n** String is too long **\n");
 exit(EXIT_FAILURE);
 }
}

/* Compare the two transformed strings and verify that */
/* the result is the same as the result from wcscoll on */
/* the original strings. */
else {
 wcscmp_result = wcscmp(w_string3, w_string4);
 if (wcscmp_result == 0 && (coll_result == 0)) {
 printf("\nReturn value from wcscoll() and return value"
 " from wcscmp() are both zero.");
 printf("\nThe program was successful\n\n");
 }
 else if ((wcscmp_result < 0) && (coll_result < 0)) {
 printf("\nReturn value from wcscoll() and return value"
 " from wcscmp() are less than zero.");
 printf("\nThe program was successful\n\n");
 }
 else if ((wcscmp_result > 0) && (coll_result > 0)) {
 printf("\nReturn value from wcscoll() and return value"
 " from wcscmp() are greater than zero.");
 printf("\nThe program was successful\n\n");
 }
 else {
 printf("*** Error **\n");
 printf("\nReturn values are not of the same type");
 }
}
}
}
}

```

Running the example program produces the following result:

```

Return value from wcscoll() and return value
 from wcscmp() are less than zero.
The program was successful

```

## wctob

---

## wctob

Determines if a wide character corresponds to a single-byte multibyte character and returns its multibyte character representation.

### Format

```
#include <stdio.h>
#include <wchar.h>
int wctob (wint_t c);
```

### Arguments

**c**  
The wide character to be converted to a single-byte multibyte character.

### Description

The `wctob` function determines whether the specified wide character corresponds to a single-byte multibyte character when in the initial shift state and, if so, returns its multibyte character representation.

### Return Values

|                  |                                                                                                            |
|------------------|------------------------------------------------------------------------------------------------------------|
| <code>x</code>   | The single-byte representation of the wide character specified.                                            |
| <code>EOF</code> | Indicates an error. The wide character specified does not correspond to a single-byte multibyte character. |

---

## wctomb

Converts a wide character to its multibyte character representation.

### Format

```
#include <stdlib.h>

int wctomb (char *s, wchar_t wchar);
```

### Arguments

**s**

A pointer to the resulting multibyte character.

**wchar**

The code for the wide character.

### Description

The `wctomb` function converts the wide character specified by *wchar* to its multibyte character representation. If *s* is `NULL`, then 0 is returned. Otherwise, the number of bytes comprising the multibyte character is returned. At most, `MB_CUR_MAX` bytes are stored in the array object pointed to by *s*.

This function is affected by the `LC_CTYPE` category of the program's current locale.

### Return Values

|    |                                                                                        |
|----|----------------------------------------------------------------------------------------|
| x  | The number of bytes comprising the multibyte character corresponding to <i>wchar</i> . |
| 0  | If <i>s</i> is <code>NULL</code> .                                                     |
| -1 | If <i>wchar</i> is not a valid character.                                              |

---

## wctrans

Returns the description of a mapping, corresponding to specified property, that can later be used in a call to `towctrans`.

### Format

```
#include <wctype.h>

wctrans_t wctrans (const char *property);
```

### Arguments

#### **property**

The name of the mapping. The following property names are defined for all locales:

- "toupper"
- "tolower"

Additional property names may also be defined in the `LC_CTYPE` category of the current locale.

### Description

The `wctrans` function constructs a value with type `wctrans_t` that describes a mapping between wide characters identified by the *property* argument.

See also `towctrans` in this section.

### Return Values

|         |                                                                                                                                                                                                                                                           |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| nonzero | According to the <code>LC_CTYPE</code> category of the current program locale, the string specified as a property argument is the name of an existing character mapping. The value returned can be used in a call to the <code>towctrans</code> function. |
| 0       | Indicates an error. The property argument does not identify a character mapping in the current program's locale.                                                                                                                                          |



## wctype

Used for defining a character class. The value returned by this function is used in calls to the `iswctype` function.

### Format

```
#include <wctype.h> (ISO C)

#include <wchar.h> (XPG4)

wctype_t wctype (const char *char_class);
```

### Arguments

#### **char\_class**

A pointer to a valid character class name.

### Description

The `wctype` function converts a valid character class defined for the current locale to an object of type `wctype_t`. The following character class names are defined for all locales:

|       |       |       |        |
|-------|-------|-------|--------|
| alnum | cntrl | lower | space  |
| alpha | digit | print | upper  |
| blank | graph | punct | xdigit |

Additional character class names may also be defined in the `LC_CTYPE` category of the current locale.

See also `iswctype` in this section.

### Return Values

|   |                                                                                                          |
|---|----------------------------------------------------------------------------------------------------------|
| x | An object of type <code>wctype_t</code> that can be used in calls to the <code>iswctype</code> function. |
| 0 | If the character class name is not valid for the current locale.                                         |

### Example

```
#include <locale.h>
#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

/* This test will set up a number of character class using wctype() */
/* and then verify whether calls to iswctype() using these classes */
/* produce the same results as calls to the is**** routines. */

main()
{
 wchar_t w_char;
 wctype_t ret_val;

 char *character = "A";

 /* Convert character to wide character format - w_char */
```

## wctype

```
if (mbtowc(&w_char, character, 1) == -1) {
 perror("mbtowc");
 exit(EXIT_FAILURE);
}

/* Check if results from iswalnum() matches check on */
/* alnum character class */

if ((iswalnum((wint_t) w_char)) &&
 (iswctype((wint_t) w_char, wctype("alnum"))))
 printf("[%C] is a member of the character class alnum\n", w_char);
else
 printf("[%C] is not a member of the character class alnum\n", w_char);

/* Check if results from iswalpha() matches check on */
/* alpha character class */

if ((iswalpha((wint_t) w_char)) &&
 (iswctype((wint_t) w_char, wctype("alpha"))))
 printf("[%C] is a member of the character class alpha\n", w_char);
else
 printf("[%C] is not a member of the character class alpha\n", w_char);

/* Check if results from iswcntrl() matches check on */
/* cntrl character class */

if ((iswcntrl((wint_t) w_char)) &&
 (iswctype((wint_t) w_char, wctype("cntrl"))))
 printf("[%C] is a member of the character class cntrl\n", w_char);
else
 printf("[%C] is not a member of the character class cntrl\n", w_char);

/* Check if results from iswdigit() matches check on */
/* digit character class */

if ((iswdigit((wint_t) w_char)) &&
 (iswctype((wint_t) w_char, wctype("digit"))))
 printf("[%C] is a member of the character class digit\n", w_char);
else
 printf("[%C] is not a member of the character class digit\n", w_char);

/* Check if results from iswgraph() matches check on */
/* graph character class */

if ((iswgraph((wint_t) w_char)) &&
 (iswctype((wint_t) w_char, wctype("graph"))))
 printf("[%C] is a member of the character class graph\n", w_char);
else
 printf("[%C] is not a member of the character class graph\n", w_char);

/* Check if results from iswlower() matches check on */
/* lower character class */

if ((iswlower((wint_t) w_char)) &&
 (iswctype((wint_t) w_char, wctype("lower"))))
 printf("[%C] is a member of the character class lower\n", w_char);
else
 printf("[%C] is not a member of the character class lower\n", w_char);

/* Check if results from iswprint() matches check on */
/* print character class */

if ((iswprint((wint_t) w_char)) &&
 (iswctype((wint_t) w_char, wctype("print"))))
 printf("[%C] is a member of the character class print\n", w_char);
else
 printf("[%C] is not a member of the character class print\n", w_char);

/* Check if results from iswpunct() matches check on */
/* punct character class */
```

```

if ((iswpunct((wint_t) w_char) &&
 (iswctype((wint_t) w_char, wctype("punct"))))
 printf("[%C] is a member of the character class punct\n", w_char);
else
 printf("[%C] is not a member of the character class punct\n", w_char);
/* Check if results from iswspace() matches check on */
/* space character class */
if ((iswspace((wint_t) w_char) &&
 (iswctype((wint_t) w_char, wctype("space"))))
 printf("[%C] is a member of the character class space\n", w_char);
else
 printf("[%C] is not a member of the character class space\n", w_char);
/* Check if results from iswupper() matches check on */
/* upper character class */
if ((iswupper((wint_t) w_char) &&
 (iswctype((wint_t) w_char, wctype("upper"))))
 printf("[%C] is a member of the character class upper\n", w_char);
else
 printf("[%C] is not a member of the character class upper\n", w_char);
/* Check if results from iswxdigit() matches check on */
/* xdigit character class */
if ((iswxdigit((wint_t) w_char) &&
 (iswctype((wint_t) w_char, wctype("xdigit"))))
 printf("[%C] is a member of the character class xdigit\n", w_char);
else
 printf("[%C] is not a member of the character class xdigit\n", w_char);
}

```

Running this example produces the following result:

```

[A] is a member of the character class alnum
[A] is a member of the character class alpha
[A] is not a member of the character class cntrl
[A] is not a member of the character class digit
[A] is a member of the character class graph
[A] is not a member of the character class lower
[A] is a member of the character class print
[A] is not a member of the character class punct
[A] is not a member of the character class space
[A] is a member of the character class upper
[A] is a member of the character class xdigit

```

## wcwidth

---

## wcwidth

Determines the number of printing positions on a display device required for the specified wide character.

### Format

```
#include <wchar.h>

int wcwidth (wchar_t wc);
```

### Arguments

**wc**  
A wide character.

### Description

The `wcwidth` function determines the number of column positions needed for the specified wide character *wc*. The value of *wc* must be a valid wide character in the current locale.

### Return Values

|    |                                                                              |
|----|------------------------------------------------------------------------------|
| x  | The number of printing positions required for <i>wc</i> .                    |
| 0  | If <i>wc</i> is a null character.                                            |
| −1 | Indicates that <i>wc</i> does not represent a valid printing wide character. |

---

## wmemchr

Locates the first occurrence of a specified wide character in an array of wide characters.

### Format

```
#include <wchar.h>

wchar_t wmemchr (const wchar_t *s, wchar_t c, size_t n);
```

### Function Variants

The `wmemchr` function has variants named `_wmemchr32` and `_wmemchr64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**s**  
A pointer to an array of wide characters to be searched.

**c**  
The wide character value to search for.

**n**  
The maximum number of wide characters in the array to be searched.

### Description

The `wmemchr` function locates the first occurrence of the specified wide character in the initial *n* wide characters of the array pointed to by *s*.

### Return Values

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| <code>x</code>    | A pointer to the first occurrence of the wide character in the array. |
| <code>NULL</code> | The specified wide character does not occur in the array.             |

## wmemcmp

---

## wmemcmp

Compares two arrays of wide characters.

### Format

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

```
int wmemcmp (const wchar_t *s1, const wchar_t *s2, size_t n);
```

### Arguments

**s1, s2**

Pointers to wide-character arrays.

**n**

The maximum number of wide characters to be compared.

### Description

The `wmemcmp` function compares the first  $n$  wide characters of the array pointed to by `s1` with the first  $n$  wide characters of the array pointed to by `s2`. The wide characters are compared not according to locale-dependent collation rules, but as integral objects of type `wchar_t`.

### Return Values

0

Arrays are equal.

Positive value

The first array is greater than the second.

Negative value

The first array is less than the second.

---

## wmemcpy

Copies a specified number of wide characters from one wide-character array to another.

### Format

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

```
wchar_t wmemcpy (wchar_t *dest, const wchar_t *source, size_t n);
```

### Function Variants

The `wmemcpy` function has variants named `_wmemcpy32` and `_wmemcpy64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**

A pointer to the destination array.

**source**

A pointer to the source array.

**n**

The number of wide characters to be copied.

### Description

The `wmemcpy` function copies *n* wide characters from the array pointed to by *source* to the array pointed to by *dest*.

### Return Value

x

The value of *dest*.

---

## wmemmove

Copies a specified number of wide characters from one wide-character array to another.

### Format

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

```
wchar_t wmemmove (wchar_t *dest, const wchar_t *source, size_t n);
```

### Function Variants

The `wmemmove` function has variants named `_wmemmove32` and `_wmemmove64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**dest**

A pointer to the destination array.

**source**

A pointer to the source array.

**n**

The number of wide characters to be moved.

### Description

The `wmemmove` function copies *n* wide characters from the location pointed to by *source* to the location pointed to by *dest*.

The `wmemmove` and `wmemcpy` routines perform the same function, except that `wmemmove` ensures that the original contents of the source array are copied to the destination array even if the two arrays overlap. Where such overlap is possible, programs that require portability should use `wmemmove`, not `wmemcpy`.

### Return Value

x

The value of *dest*.



---

## wmemset

Sets a specified value to a specified number of wide characters in an array of wide characters.

### Format

```
#include <wchar.h>

wchar_t wmemset (wchar_t *s, wchar_t c, size_t n);
```

### Function Variants

The `wmemset` function has variants named `_wmemset32` and `_wmemset64` for use with 32-bit and 64-bit pointer sizes, respectively. See Section 1.10 for more information on using pointer-size-specific functions.

### Arguments

**s**

A pointer to the array of wide characters.

**c**

The value to be placed in the first  $n$  wide characters of the array.

**n**

The number of wide characters to be set to the specified value  $c$ .

### Description

The `wmemset` function copies the value of  $c$  into each of the first  $n$  wide characters of the array pointed to by  $s$ .

### Return Value

**x**

The value of  $s$ .

## wprintf

---

## wprintf

Performs formatted output from the standard output (stdout). See Chapter 2 for information on format specifiers.

### Format

```
#include <wchar.h>

int wprintf (const wchar_t *format, . . .);
```

### Arguments

#### **format**

A pointer to a wide-character string containing the format specifications. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

. . .

Optional expressions whose resultant types correspond to conversion specifications given in the format specification.

If no conversion specifications are given, the output sources can be omitted. Otherwise, the function calls must have exactly as many output sources as there are conversion specifications, and the conversion specifications must match the types of the output sources.

Conversion specifications are matched to output sources in left-to-right order. Excess output pointers, if any, are ignored.

### Description

The `wprintf` function is equivalent to the `fwprintf` function with the `stdout` argument interposed before the `wprintf` arguments.

### Return Values

|   |                                        |
|---|----------------------------------------|
| n | The number of wide characters written. |
|---|----------------------------------------|

Negative value

Indicates an error. The function sets `errno` to one of the following:

- `EILSEQ` – Invalid character detected.
- `EINVAL` – Insufficient arguments.
- `ENOMEM` – Not enough memory available for conversion.
- `ERANGE` – Floating-point calculations overflow.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This might indicate that conversion to a numeric value failed because of overflow.

The function can also set `errno` to the following as a result of errors returned from the I/O subsystem:

- `EBADF` – The file descriptor is not valid.
- `EIO` – I/O error.
- `ENOSPC` – No free space on the device containing the file.
- `ENXIO` – Device does not exist.
- `EPIPE` – Broken pipe.
- `ESPIPE` – Illegal seek in a file opened for append.
- `EVMSError` – Nontranslatable VMS error. `vaxc$errno` contains the VMS error code. This indicates that an I/O error occurred for which there is no equivalent C error code.

## wrapok

---

## wrapok

In the UNIX system environment, allows the wrapping of a word from the right border of the window to the beginning of the next line. This routine is provided only for UNIX software compatibility and serves no function in the OpenVMS environment.

### Format

```
#include <curses.h>

wrapok (WINDOW *win, bool boolf);
```

### Arguments

#### **win**

A pointer to the window.

#### **boolf**

A Boolean TRUE or FALSE value. If *boolf* is FALSE, scrolling is not allowed. This is the default setting. The bool type is defined in the <curses.h> header file as follows:

```
#define bool int
```

---

## write

Writes a specified number of bytes from a buffer to a file.

### Format

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

```
ssize_t write (int file_desc, void *buffer, size_t nbytes); (ISO POSIX-1)
```

```
int write (int file_desc, void *buffer, int nbytes); (Compatability)
```

### Arguments

**file\_desc**

A file descriptor that refers to a file currently opened for writing or updating.

**buffer**

The address of contiguous storage from which the output data is taken.

**nbytes**

The maximum number of bytes involved in the write operation.

### Description

If the write is to an RMS record file and the buffer contains embedded new-line characters, more than one record may be written to the file. Even if there are no embedded new-line characters, if *nbytes* is greater than the maximum record size for the file, more than one record will be written to the file. The write function always generates at least one record.

If the write is to a mailbox and the third argument, *nbytes*, specifies a length of 0, an end-of-file message is written to the mailbox. This occurs for mailboxes created by the application using SYS\$CREMBX, but not for mailboxes created to implement POSIX pipes. For more information, see Chapter 5.

### Return Values

x

The number of bytes written.

−1

Indicates errors, including undefined file descriptors, illegal buffer addresses, and physical I/O errors.

## writev

---

## writev

Writes to a file.

### Format

```
#include <unistd.h>

ssize_t writev (int filides, const struct iovec *iov, int iovcnt);
```

### Argument

#### **filedes**

A file descriptor that refers to a file currently opened for writing or updating.

#### **iov**

Array of iovec structures from which the output data is gathered.

#### **iovcnt**

The number of buffers specified by the members of the *iov* array.

### Description

The `writev` function is equivalent to `write` but gathers the output data from the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*−1]. The *iovcnt* argument is valid if greater than 0 and less than or equal to {IOV\_MAX}, defined in <limits.h>.

Each iovec entry specifies the base address and length of an area in memory from which data should be written. The `writev` function writes a complete area before proceeding to the next.

If *filedes* refers to a regular file and all of the *iov\_len* members in the array pointed to by *iov* are 0, `writev` returns 0 and has no other effect.

For other file types, the behavior is unspecified.

If the sum of the *iov\_len* values is greater than SSIZE\_MAX, the operation fails and no data is transferred.

Upon successful completion, `writev` returns the number of bytes actually written. Otherwise, it returns a value of −1, the file pointer remains unchanged, and `errno` is set to indicate an error.

### Return Values

|   |                              |
|---|------------------------------|
| x | The number of bytes written. |
|---|------------------------------|

-1

Indicates an error. The file times do not change, and the function sets `errno` to one of the following values:

- `EBADF` – The *files* argument is not a valid file descriptor open for writing.
- `EINTR` – The write operation was terminated due to the receipt of a signal, and no data was transferred.
- `EINVAL` – The sum of the *iov\_len* values in the *iov* array would overflow an `ssize_t`, or the *iovcnt* argument was less than or equal to 0, or greater than `{IOV_MAX}`.
- `EIO` – A physical I/O error has occurred.
- `ENOSPC` – There was no free space remaining on the device containing the file.
- `EPIPE` – An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A `SIGPIPE` signal will also be sent to the thread.

---

## wscanf

Reads input from the standard input (stdin) under control of the wide-character format string.

### Format

```
#include <wchar.h>

int wscanf (const wchar_t *format, ...);
```

### Arguments

#### **format**

A pointer to a wide-character string containing the format specifications. For more information about format and conversion specifications and their corresponding arguments, see Chapter 2.

...

Optional expressions whose results correspond to conversion specifications given in the format specification.

If no conversion specifications are given, you can omit the input pointers. Otherwise, the function calls must have exactly as many input pointers as there are conversion specifications, and the conversion specifications must match the types of the input pointers.

Conversion specifications are matched to input sources in left-to-right order. Excess input pointers, if any, are ignored.

### Description

The `wscanf` function is equivalent to the `fwscanf` function with the `stdin` arguments interposed before the `wscanf` arguments.

### Return Values

|     |                                                                                                                                     |
|-----|-------------------------------------------------------------------------------------------------------------------------------------|
| n   | The number of input items assigned. The number can be less than provided for, even zero, in the event of an early matching failure. |
| EOF | Indicates an error. An input failure occurred before any conversion.                                                                |



---

**y0, y1, yn** *(Alpha only)*

Compute Bessel functions of the second kind.

**Format**

```
#include <math.h>
double y0 (double x);
float y0f (float x);
long double y0l (long double x);
double y1 (double x);
float y1f (float x);
long double y1l (long double x);
double yn (int n, double x);
float ynf (int n, float x);
long double ynl (int n, long double x);
```

**Arguments**

**x**  
A positive, real value.

**n**  
An integer.

**Description**

The y0 functions return the value of the Bessel function of the second kind of order 0.

The y1 functions return the value of the Bessel function of the second kind of order 1.

The yn functions return the value of the Bessel function of the second kind of order  $n$ .

**Return Values**

|           |                                                            |
|-----------|------------------------------------------------------------|
| x         | The relevant Bessel value of $x$ of the second kind.       |
| −HUGE_VAL | The $x$ argument is 0.0; errno is set to ERANGE.           |
| NaN       | The $x$ argument is negative or NaN; errno is set to EDOM. |
| 0         | Underflow occurred; errno is set to ERANGE.                |
| HUGE_VAL  | Overflow occurred; errno is set to ERANGE.                 |



---

## Compaq C Socket Routine Reference

This appendix describes the socket routines used for writing Internet application programs for the *TCP/IP Services for OpenVMS* product or other implementations of the TCP/IP protocol.

For a description of Internet details, such as protocols, protocol types, and sockets, see the *TCP/IP Services for OpenVMS* product documentation.

### A.1 Porting Considerations

This section contains information that you should consider when writing Internet application programs for the *TCP/IP Services for OpenVMS*. These considerations will help to make your programs more portable.

#### A.1.1 Calling an IPC Routine from an AST State

Calls to various Interprocess Communication (IPC) routines use a static area within which they return information. The OpenVMS environment allows an asynchronous system trap (AST) routine to interrupt an IPC routine during its execution. In addition, the ASTs of more privileged modes can interrupt ASTs of less privileged modes. Therefore, caution needs to be observed when calling an IPC routine from an AST state, while a similar IPC routine is being called from a non-AST state or a less privileged mode.

The IPC routines that use a static area are:

- gethostbyaddr
- gethostbyname
- getnetbyaddr
- getnetbyname

In VMS Version 5.2, sockets should not be created or destroyed within ASTs.

#### A.1.2 Calling from KERNEL or EXEC Modes

Several IPC routines access files in order to retrieve their information. These routines should not be called from either the KERNEL or EXEC modes when ASTs are disabled. These IPC routines are:

- gethostbyaddr
- gethostbyname
- getnetbyaddr
- getnetbyname

### A.1.3 Event Flags

IPC routines may use event flags during their operation. The event flags are assigned by using the library routine LIB\$GET\_EF and are released when the routine no longer needs them.

### A.1.4 Suppressing *Compaq C* Compilation Warnings

Certain parameters to the IPC routines may require type casting to suppress *Compaq C* compilation warnings. Type casting is required because of parameter prototyping, which the *Compaq C* header (.h) files have in order to be ANSI compliant. These header files are unlike UNIX header files, whose IPC routines are not parameter prototyped.

### A.1.5 Header Files

It is acceptable to include header files on an OpenVMS system without using angle brackets (< >) or double quotes (" "). For example, #include types. This form of the #include preprocessor directive is possible on OpenVMS systems because all header files are located in a text library in SYS\$LIBRARY. On UNIX systems, however, header files must be specified with angle brackets (< >) or double quotes (" ") and any subdirectories that are needed to locate a header file. For example, to include the header file socket.h, use the following form of the #include directive:

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

## A.2 *Compaq C* Structures

The socket routines make use of several *Compaq C* structures. Table A–1 lists these structures and the header files in which they are defined.

**Table A–1 Structures Used by Socket Routines**

| Structure Name | Header File |
|----------------|-------------|
| in_addr        | <in.h>      |
| sock_addr_in   | <in.h>      |
| hostent        | <netdb.h>   |
| netent         | <netdb.h>   |
| servent        | <netdb.h>   |
| linger         | <socket.h>  |
| sock_addr      | <socket.h>  |
| msghdr         | <socket.h>  |
| timeval        | <socket.h>  |
| iovec          | <uio.h>     |

## A.3 Internet Protocols

The Internet protocol family is a collection of protocols layered on the Internet Protocol (IP) transport layer, and using the Internet address format. This section describes the Transmission Control Protocol and User Datagram Protocol.

### A.3.1 Transmission Control Protocol

The Transmission Control Protocol (TCP) provides a reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK\_STREAM abstraction. TCP uses the standard Internet address format and, in addition, provides a per host collection of *port addresses*. Thus, each address is composed of an Internet address specifying the host and network with a specific TCP port on the host identifying the peer entity.

Sockets utilizing the TCP protocol are either *active* or *passive*. Active sockets initiate connections to passive sockets. By default, TCP sockets are created active; to create a passive socket the listen system call must be used after binding the socket with the bind system call. Only passive sockets may use the accept call to accept incoming connections. Only active sockets may use the connect call to initiate connections.

Passive sockets may *underspecify* their location to match incoming connection requests from multiple networks. This technique, called *wildcard addressing*, allows a single server to provide service to clients on multiple networks. To create a socket that listens to all hosts on any network, the Internet address INADDR\_ANY must be bound. The TCP port must be specified at this time. If the Internet address is not INADDR and the port is not specified, the system will assign a port. Once a connection has been established, the socket's address is fixed by the peer entity's location. The address assigned to the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity's network.

TCP supports one socket option that is set with setsockopt and tested with getsockopt. Under most circumstances, TCP sends data when it is presented; when outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgment is received. For a small number of clients, such as window systems that send a stream of mouse events that receive no replies, this packetization may cause significant delays. Therefore, TCP provides a Boolean option, TCP\_NODELAY (from <netinet/tcp.h>), to defeat this algorithm. The option level for the setsockopt call is the protocol number for TCP, which is available from getprotobyname.

### A.3.2 User Datagram Protocol

User Datagram Protocol (UDP) is a simple, unreliable datagram protocol used to support the SOCK\_DGRAM abstraction for the Internet protocol family. UDP sockets are connectionless and are normally used with the sendto and recvfrom calls, though the connect call may also be used to fix the destination for future packets (in which case the recv or read or write system calls may be used).

UDP address formats are identical to those used by TCP. In particular, UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port space is separate from the TCP port space (for example, a UDP port may not be connected to a TCP port). Also, broadcast packets may be sent (assuming the underlying network supports this) by using a reserved *broadcast address*; this address is network interface dependent. The SO\_BROADCAST option must be set on the socket and the process must have the SYSPRV or BYPASS privilege for broadcasting to succeed.

## A.4 errno Values

errno is an external variable whose value is set whenever an error occurs during a call to any of the *Compaq C* RTL routines. You can use this value to obtain a more detailed description of the error. errno is not cleared on successful calls, so its value should be checked only when an error has been indicated.

Most calls to the *Compaq C* RTL routines have one or more returned values. Any error condition is indicated by an otherwise impossible return value. This is almost always `-1`; the individual routine descriptions specify the details.

All return codes and values from routines are of type `int` unless otherwise noted. An error number is also made available in the external variable `errno`, which is not cleared on successful calls. The `errno` values may be translated to a message, similar to that found in UNIX systems, by using the `perror` routine. `vaxc$errno` may also be returned as an error.

---

### NOTE

---

The notation [...] is used in this manual to denote an `errno` error.

---

Table A-2 lists the `errno` values.

**Table A-2** `errno` Values

| Value           | Meaning                                                                                                                                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EINPROGRESS     | Operation now in progress<br>An operation that takes a long time to complete, such as <code>connect</code> , was attempted on a nonblocking object.                                                                          |
| EALREADY        | Operation already in progress<br>An operation was attempted on a nonblocking object that already had an operation in progress.                                                                                               |
| ENOTSOCK        | Socket operation on a nonsocket                                                                                                                                                                                              |
| EDESTADDRREQ    | Destination address required<br>A required address was omitted from an operation on a socket.                                                                                                                                |
| EMSGSIZE        | Message too long<br>A message sent on a socket was larger than the internal message buffer.                                                                                                                                  |
| EPROTOTYPE      | Protocol wrong type for socket<br>A protocol was specified that does not support the semantics of the socket type requested. For example, you cannot use the ARPA Internet UDP protocol with type <code>SOCK_STREAM</code> . |
| ENOPROTOOPT     | Protocol not available<br>A bad option was specified in a <code>getsockopt</code> or <code>setsockopt</code> call.                                                                                                           |
| EPROTONOSUPPORT | Protocol not supported                                                                                                                                                                                                       |

(continued on next page)

**Table A–2 (Cont.) errno Values**

| Value           | Meaning                                                                                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 | The protocol has not been configured into the system or no implementation for it exists.                                                                                 |
| ESOCKTNOSUPPORT | Socket type not supported                                                                                                                                                |
|                 | The support for the socket type has not been configured into the system or no implementation for it exists.                                                              |
| EOPNOTSUPP      | Error-operation not supported                                                                                                                                            |
|                 | For example, trying to accept a connection on a datagram socket.                                                                                                         |
| EPFNOSUPPORT    | Protocol family not supported                                                                                                                                            |
|                 | The protocol family has not been configured into the system or no implementation for it exists.                                                                          |
| EAFNOSUPPORT    | Address family not supported by protocol family                                                                                                                          |
|                 | An address incompatible with the requested protocol was used.                                                                                                            |
| EADDRINUSE      | Address already in use                                                                                                                                                   |
|                 | Each address can be used only once.                                                                                                                                      |
| EADDRNOTAVAIL   | Cannot assign requested address                                                                                                                                          |
|                 | Normally, results from an attempt to create a socket with an address not on this machine.                                                                                |
| ENETDOWN        | Network is down                                                                                                                                                          |
|                 | A socket operation encountered a dead network.                                                                                                                           |
| ENETUNREACH     | Network is unreachable                                                                                                                                                   |
|                 | A socket operation was attempted to an unreachable network.                                                                                                              |
| ENETRESET       | Network dropped connection on reset                                                                                                                                      |
|                 | The host you were connected to crashed and rebooted.                                                                                                                     |
| ECONNABORTED    | Software caused connection abort                                                                                                                                         |
|                 | A connection abort was caused internal to your host machine.                                                                                                             |
| ECONNRESET      | Connection reset by peer                                                                                                                                                 |
|                 | A connection was forcibly closed by a peer. This usually results from the peer executing a shutdown call.                                                                |
| ENOBUFS         | No buffer space available                                                                                                                                                |
|                 | An operation on a socket or pipe was not performed because the system lacked sufficient buffer space.                                                                    |
| EISCONN         | Socket is already connected                                                                                                                                              |
|                 | A connect request was made on an already connected socket; or, a sendto or sendmsg request on a connected socket specified a destination other than the connected party. |
| ENTOTCONN       | Socket is not connected                                                                                                                                                  |

(continued on next page)

**Table A–2 (Cont.) errno Values**

| Value        | Meaning                                                                                                                                                                                                                                                                                                                                                                         |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | Request to send or receive data was disallowed because the socket is not connected.                                                                                                                                                                                                                                                                                             |
| ESHUTDOWN    | Cannot send after socket shutdown                                                                                                                                                                                                                                                                                                                                               |
|              | A request to send data was disallowed because the socket had already been shut down with a previous shutdown call.                                                                                                                                                                                                                                                              |
| ETOOMANYREFS | Too many references: cannot splice                                                                                                                                                                                                                                                                                                                                              |
| ETIMEDOUT    | Connection timed out                                                                                                                                                                                                                                                                                                                                                            |
|              | A <code>connect</code> request failed because the connected party did not properly respond after a period of time. (The timeout period is dependent on the communication protocol.) A <code>connect</code> request or remote file operation failed because the connected party did not properly respond after a period of time that is dependent on the communication protocol. |
| ECONNREFUSED | Connection refused                                                                                                                                                                                                                                                                                                                                                              |
|              | No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host.                                                                                                                                                                                                      |
| ELOOP        | Too many levels of symbolic links                                                                                                                                                                                                                                                                                                                                               |
|              | A pathname lookup involved more than eight symbolic links.                                                                                                                                                                                                                                                                                                                      |
| ENAMETOOLONG | File name too long                                                                                                                                                                                                                                                                                                                                                              |
|              | A component of a pathname exceeded 255 characters, or an entire pathname exceeded 1023 characters.                                                                                                                                                                                                                                                                              |
| EHOSTDOWN    | Host is down                                                                                                                                                                                                                                                                                                                                                                    |
|              | A socket operation failed because the destination host was down.                                                                                                                                                                                                                                                                                                                |
| EHOSTUNREACH | No route to host                                                                                                                                                                                                                                                                                                                                                                |
|              | A socket operation was attempted to an unreachable host.                                                                                                                                                                                                                                                                                                                        |
| EVMSError    | OpenVMS system-specific error code that is nontranslatable                                                                                                                                                                                                                                                                                                                      |

## A.5 h\_errno Values

The external integer `h_errno` is available only with OpenVMS Version 7.0, and is set only by a 4.4BSD TCP/IP interface. Specifically, The `gethostbyname` and `gethostbyaddr` functions require 4.4BSD semantics to set `h_errno`. The `gethostbyname` and `gethostbyaddr` functions indicate an error condition by returning a null pointer and setting the external integer `h_errno` to indicate the error return status.

When `gethostbyname` or `gethostbyaddr` returns an error status, `h_errno`, which is very similar to `errno`, can be checked to determine whether the error is the result of a temporary failure or an invalid or unknown host.



Use the `herror` routine to print the error message describing the failure. If the argument string to `herror` is not `NULL`, it is printed, followed by a colon (:) and a space. The error message is printed with a trailing new-line character.

---

**Note**

---

The `h_errno` error codes are *not* available with the `errno` variable.

---

The `<netdb.h>` header file declares `h_errno` on a per-thread basis as:

```
#define h_errno (*decc$h_errno_get_addr())
```

The `<netdb.h>` header file also symbolically defines the error code values that `h_errno` can accept, as follows:

|                             |                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>HOST_NOT_FOUND</code> | No such host is known.                                                                                                                                                                                                                                                                                                                |
| <code>TRY_AGAIN</code>      | Usually a temporary error that means the local server did not receive a response from an authoritative server. A retry at some later time can succeed.                                                                                                                                                                                |
| <code>NO_RECOVERY</code>    | An unexpected server failure is encountered. This is a nonrecoverable error.                                                                                                                                                                                                                                                          |
| <code>NO_DATA</code>        | The requested name is valid but does not have an IP address; this is not a temporary error. The name is known to the name server but there is no address associated with this name. Another type of request to the name server using this domain name results in an answer; for example, a mail-forwarder registered for this domain. |
| <code>NO_ADDRESS</code>     | No address; look for MX record                                                                                                                                                                                                                                                                                                        |

Like `errno`, the value of `h_errno` is zero at program startup. Checking `h_errno` is valid only when a failure status is returned by a *Compaq C* RTL routine that is defined to set it.

## A.6 Relationship Between `errno` and `h_errno`

Since the routines that set `h_errno` can also set `errno`, this section explains the relationship between the two variables.

A given routine failure does not set both `h_errno` and `errno` for the same failure. A routine failure sets either `h_errno` or `errno`, depending on which is appropriate for the failing condition.

For example, consider the following program:

```
/* Demonstrate relationship between errno and h_errno */
/* from Appendix A of the reference manual */
#include <errno.h>
#include <stdio.h>
#include <netdb.h>

main()
{
 static char hostname[256];
 struct hostent *hostentp;

```

```

 errno = 0;
 h_errno = 0;
 if ((hostentptr = gethostbyname("hndy")) == NULL) {
 printf("unknown host name errno is: %d h_errno is: %d\n", errno, h_errno);
 perror("p_gethostbyname");
 perror("h_gethostbyname");
 }

 errno = 0;
 h_errno = 0;
 if ((hostentptr = gethostbyname(0)) == NULL) {
 printf("illegal host name errno is: %d h_errno is: %d\n", errno, h_errno);
 perror("p_gethostbyname");
 perror("h_gethostbyname");
 }
}

```

In the first call to `gethostbyname`, the host *name* parameter "hndy" does not represent a known host. In this case, `gethostbyname` sets `h_errno` to `HOST_NOT_FOUND`, but does not set `errno`.

The call to `perror` in this example would output:

```
p_gethostbyname: Error 0
```

The call to `herror` in this example would print a message describing the failure:

```
h_gethostbyname: Unknown
```

In the second call to `gethostbyname`, the host *name* parameter is 0 (zero), an invalid argument. In this case, `gethostbyname` sets `errno` to `EINVAL`, but does not set `h_errno`.

A call to `perror` would print a message describing the failure:

```
p_gethostbyname: Invalid
```

A call to `herror` would print:

```
h_gethostbyname: Error 0
```

## A.7 TCP/IP Interface Enhancements

The *Compaq C* RTL provides the language bindings for a 4.4BSD-compatible TCP/IP implementation. The underlying 4.4BSD semantics for the specific TCP/IP routines are the responsibility of the TCP/IP backend vendors.

With 4.4BSD semantics, `gethostbyname` and `gethostbyaddr` set the external variable `h_errno`. Other routines (see Table A-3) deal with 4.4BSD versions of the `sockaddr` and `msghdr` data structures. See the `<socket.h>` header file for a description of these structures.

The TCP/IP backend routines need to know the following:

- Whether `gethostbyname` and `gethostbyaddr` should set `h_errno`.
- What type of `sockaddr` and `msghdr` structures the *Compaq C* RTL is passing them—those defined by 4.4BSD systems, or those defined by pre-4.4BSD systems).

This information is controlled by the `_SOCKADDR_LEN` feature-test macro (see Section 1.5).

Compiling with `_SOCKADDR_LEN` defined will enable the entry points that have 4.4BSD semantics. See Table A-3. Otherwise, entry points expecting pre-4.4BSD semantics are enabled.

**Table A-3 4.4BSD Entry Points**

|                              |                                    |                               |
|------------------------------|------------------------------------|-------------------------------|
| <code>__bsd44_accept</code>  | <code>__bsd44_gethostbyaddr</code> | <code>__bsd44_recvfrom</code> |
| <code>__bsd44_bind</code>    | <code>__bsd44_gethostbyname</code> | <code>__bsd44_recvmsg</code>  |
| <code>__bsd44_connect</code> | <code>__bsd44_getpeername</code>   | <code>__bsd44_sendto</code>   |
|                              | <code>__bsd44_getsockname</code>   | <code>__bsd44_sendmsg</code>  |

## A.8 Summary of Socket Routines

This section groups and briefly describes the socket routines. For complete descriptions, see the routine reference section that follows in this appendix.

### A.8.1 Basic Communication Routines

Table A-4 lists the basic communication routines that make up the building blocks of Internet programs.

**Table A-4 Basic Communication Routines**

| Routine               | Description                                                             |
|-----------------------|-------------------------------------------------------------------------|
| <code>accept</code>   | Accepts a connection on a socket.                                       |
| <code>bind</code>     | Binds a name to a socket.                                               |
| <code>close</code>    | Closes a connection and deletes a socket descriptor.                    |
| <code>connect</code>  | Initiates a connection on a socket.                                     |
| <code>ioctl</code>    | Controls socket operations only.                                        |
| <code>listen</code>   | Sets the maximum limit of outstanding connection requests for a socket. |
| <code>read</code>     | Reads bytes from a file or socket and places them into a buffer.        |
| <code>readv</code>    | Not implemented.                                                        |
| <code>recv</code>     | Receives bytes from a socket and places them into a buffer.             |
| <code>recvfrom</code> | Receives bytes for a socket from any source.                            |
| <code>recvmsg</code>  | Receives bytes from a socket and places them into scattered buffers.    |
| <code>select</code>   | Allows the polling or checking of a group of sockets.                   |
| <code>send</code>     | Sends bytes through a socket to a connected peer.                       |
| <code>sendmsg</code>  | Sends gathered bytes through a socket to any other socket.              |
| <code>sendto</code>   | Sends bytes through a socket to any other socket.                       |
| <code>shutdown</code> | Shuts down all or part of a bidirectional socket.                       |
| <code>socket</code>   | Creates an endpoint for communication by returning a socket descriptor. |
| <code>write</code>    | Writes bytes from a buffer to a file or socket.                         |
| <code>writev</code>   | Not implemented.                                                        |

## A.8.2 Auxiliary Communication Routines

Table A–5 lists the auxiliary communication routines. These routines are used to provide information about a socket and to set the options on a socket.

**Table A–5 Auxiliary Communication Routines**

| Routine     | Description                                |
|-------------|--------------------------------------------|
| getpeername | Returns the name of the connected peer.    |
| getsockname | Returns the name associated with a socket. |
| getsockopt  | Returns the options set on a socket.       |
| setsockopt  | Sets options on a socket.                  |

## A.8.3 h\_errno Support Routines

Table A–6 lists the h\_errno support routines. These routines map and write error-message strings generated by the external integer h\_errno.

**Table A–6 Supported h\_errno Routines**

| Routine   | Description                                  |
|-----------|----------------------------------------------|
| herror    | Writes a message explaining a routine error. |
| hstrerror | Accesses message explaining routine errors.  |

## A.8.4 Communication Support Routines

Table A–7 lists the communication support routines. These routines perform operations such as searching databases, converting byte order of network and host addresses, reading records, and returning Internet addresses.

**Table A–7 Supported Communication Routines**

| Routine       | Description                                                                                                                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| decc\$get_sdc | Returns the socket device's OpenVMS I/O channel associated with a socket descriptor. vaxc\$get_sdc and decc\$get_sdc are synonyms for the same routine. |
| endhostent    | Closes retrieval of network host entries and closes the network host file.                                                                              |
| endnetent     | Closes the networks database file.                                                                                                                      |
| endprotoent   | Closes the protocols database file.                                                                                                                     |
| endservent    | Closes the network services database file.                                                                                                              |
| gethostaddr   | Returns the standard host address for the processor.                                                                                                    |
| gethostbyaddr | Searches the host database for a host record with a given address.                                                                                      |
| gethostbyname | Searches the host database for a host record with a given name or alias.                                                                                |
| gethostname   | Returns the name of the current host.                                                                                                                   |

(continued on next page)

**Table A-7 (Cont.) Supported Communication Routines**

| <b>Routine</b>   | <b>Description</b>                                                                                                            |
|------------------|-------------------------------------------------------------------------------------------------------------------------------|
| gethostent       | Opens the network host entry by name from the network host database file.                                                     |
| getnetbyaddr     | Searches the network database for a network record with a given address.                                                      |
| getnetbyname     | Searches the network database for a network record with a given name or alias.                                                |
| getnetent        | Gets a network file entry from the networks database file.                                                                    |
| getprotobyname   | Searches the protocols database until a matching protocol name is found or until EOF is encountered.                          |
| getprotobynumber | Searches the protocols database until a matching protocol number is found or until EOF is encountered.                        |
| getprotoent      | Gets a protocol database entry from the protocols database file.                                                              |
| getservbyname    | Gets information on the named service from the network services database.                                                     |
| getservbyport    | Gets information on the named port from the network services database.                                                        |
| getservent       | Gets a services file entry from the network services database file.                                                           |
| hostalias        | Searches for host aliases associated with a name.                                                                             |
| htonl            | Converts longwords from network to host byte order.                                                                           |
| htons            | Converts short integers from network to host byte order.                                                                      |
| inet_addr        | Converts Internet addresses in text form into numeric Internet addresses.                                                     |
| inet_lnaof       | Returns the local network address portion of an Internet address.                                                             |
| inet_makeaddr    | Returns an Internet address given a network address and a local address on that network.                                      |
| inet_netof       | Returns the Internet network address portion of an Internet address.                                                          |
| inet_network     | Converts a null-terminated text string representing an Internet network address into a network address in network byte order. |
| inet_ntoa        | Converts an Internet address into an ASCII (null-terminated) string.                                                          |
| ntohl            | Converts longwords from host to network byte order.                                                                           |
| ntohs            | Converts short integers from host to network byte order.                                                                      |
| sethostent       | Opens, rewinds, and closes the network host database file.                                                                    |
| setnetent        | Opens, rewinds, and closes the networks database file.                                                                        |
| setprotoent      | Opens, rewinds, and closes the protocols database file.                                                                       |
| setservent       | Opens, rewinds, and closes the network services database file.                                                                |
| decc\$socket_fd  | Returns the socket descriptor associated with a Socket Device Channel for direct use with the <i>Compaq C</i> RTL.            |

(continued on next page)

**Table A–7 (Cont.) Supported Communication Routines**

| Routine       | Description                                                                                                                                             |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| vaxc\$get_sdc | Returns the socket device's OpenVMS I/O channel associated with a socket descriptor. vaxc\$get_sdc and decc\$get_sdc are synonyms for the same routine. |

### A.8.5 IPv6 Routines

Table A–8 lists the IPv6 socket routines supported by the C RTL. For more information on IPv6 features and routines, see the TCP/IP Services for OpenVMS product documentation.

**Table A–8 IPv6 Routines**

| Routine          | Description                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------------------|
| if_freenameindex | Frees dynamic memory allocated by if_nameindex to the array of interface names and indexes.                   |
| if_indextoname   | Maps an interface index to its corresponding name.                                                            |
| if_nameindex     | Returns an array of all interface names and indexes.                                                          |
| if_nametoindex   | Maps an interface name to its corresponding index.                                                            |
| freeaddrinfo     | Returns addrinfo structures and dynamic storage to the system.                                                |
| freehostent      | Deprecated routine. Replace with freeaddrinfo.                                                                |
| gai_strerror     | Describes an error value for the getaddrinfo and getnameinfo functions.                                       |
| getaddrinfo      | Protocol-independent function for mapping names to addresses.                                                 |
| getipnodebyname  | Deprecated routine. Replace with getnameinfo.                                                                 |
| getipnodebyaddr  | Deprecated routine. Replace with getaddrinfo.                                                                 |
| getnameinfo      | Protocol-independent function for mapping addresses to names.                                                 |
| inet_aton        | Converts a string to an Internet address stored in a structure. Obsoletes the inet_addr function.             |
| inet_ntop        | Converts a numeric address to a text string suitable for presentation.                                        |
| inet_pton        | Converts an address in its standard text presentation form to its numeric binary form, in network byte order. |

---

## accept

Accepts a connection on a socket.

### Format

```
#include <socket.h>

int accept (int s, struct sockaddr *addr, int *addrlen); (_DECC_V4_SOURCE)

int accept (int s, struct sockaddr *addr, size_t *addrlen); (not _DECC_V4_SOURCE)
```

### Routine Variants

The `accept` routine has a variant named `__bsd44_accept`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

#### **s**

A socket descriptor that has been returned by `socket`, subsequently bound to an address with `bind`, and that is listening for connections after a `listen`.

#### **addr**

A result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the structure to which the address parameter points is determined by the domain in which the communication is occurring. This version of *Compaq C* supports only the Internet domain (`AF_INET`).

#### **addrlen**

A value-result parameter; it should initially contain the size of the structure pointed to by `addr`. On return it will contain the actual length, in bytes, of the structure that has been filled in by the communication layer. See `<socket.h>` for a description of the `sockaddr` structure.

### Description

The `accept` routine completes the first connection on the queue of pending connections, creates a new socket with the same properties as `s`, and allocates and returns a new descriptor for the socket. If no pending connections are present on the queue, and the socket is not marked as nonblocking, `accept` blocks the caller until a connection request is present. If the socket is marked nonblocking by using a `setsockopt` call and no pending connections are present on the queue, `accept` returns an error. The accepted socket may not be used to accept connections. The original socket `s` remains open (listening) for other connection requests. This call is used with connection-based socket types, currently with `SOCK_STREAM`.

It is possible to select a socket for the purposes of performing an `accept` by selecting it for read.

See also `bind`, `connect`, `listen`, `select`, and `socket` in this appendix.

## accept

### Return Values

x

A nonnegative integer that is a descriptor for the accepted socket.

−1

Indicates an error; `errno` is set to one of the following:

- `EBADF` – The socket descriptor is invalid.
- `ENOTSOCK` – The socket descriptor references a file, not a socket.
- `EOPNOTSUPP` – The reference socket is not of type `SOCK_STREAM`.
- `EFAULT` – The *addr* parameter is not in a writable part of the user address space.
- `EWOULDBLOCK` – The socket is marked nonblocking and no connections are present to be accepted.



---

## bind

Binds a name to a socket.

### Format

```
#include <socket.h>
```

```
int bind (int s, struct sockaddr *name, int namelen); (_DECC_V4_SOURCE)
```

```
int bind (int s, const struct sockaddr *name, size_t namelen); (not _DECC_V4_SOURCE)
```

### Routine Variants

The bind routine has a variant named `__bsd44_bind`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

**s**

A socket descriptor that has been created with `socket`.

**name**

Address of a structure used to assign a name to the socket in the format specific to the family (AF\_INET) socket address. See `<socket.h>` for a description of the `sockaddr` structure.

**namelen**

The size, in bytes, of the structure pointed to by *name*.

### Description

The bind routine assigns a name to an unnamed socket. When a socket is created with `socket` it exists in a name space (address family) but has no name assigned. The bind routine requests that a name be assigned to the socket.

See also `connect`, `getsockname`, `listen`, and `socket` in this appendix.

### Return Values

0

Indicates success.

## bind

– 1

Indicates an error; `errno` is set to one of the following values:

- `EBADF` – The socket descriptor is invalid.
- `ENOTSOCK` – The socket descriptor references a file, not a socket.
- `EADDRNOTAVAIL` – The specified address is not available from the local machine.
- `EADDRINUSE` – The specified Internet address and ports are already in use.
- `EINVAL` – The socket is already bound to an address.
- `EACCESS` – The requested address is protected, and the current user has inadequate permission to access it.
- `EFAULT` – The *name* parameter is not a valid part of the user address space.

---

## close

Closes a connection and deletes a socket descriptor.

### Format

```
#include <unistd.h>

int close (s);
```

### Argument

**s**  
A socket descriptor.

### Description

The `close` routine deletes a descriptor from the per-process object reference table. If this is the last reference to the underlying object, then it will be deactivated.

See also `accept`, `socket`, and `write` in this appendix.

### Return Values

|    |                                                                                                         |
|----|---------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                      |
| −1 | Indicates an error; <code>errno</code> is set to <code>EBADF</code> (The socket descriptor is invalid.) |

## connect

---

## connect

Initiates a connection on a socket.

### Format

```
#include <socket.h>
```

```
int connect (int s, struct sockaddr *name, int namelen); (_DECC_V4_SOURCE)
```

```
int connect (int s, const struct sockaddr *name, size_t namelen); (not _DECC_V4_SOURCE)
```

### Routine Variants

The connect routine has a variant named `__bsd44_connect`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

#### **s**

A socket descriptor that has been created with `socket`.

#### **name**

The address of a structure that specifies the name of the remote socket in the format specific to the address family (`AF_INET`).

#### **namelen**

The size, in bytes, of the structure pointed to by *name*.

### Description

If *s* is a socket descriptor of type `SOCK_DGRAM`, then this call permanently specifies the peer to which data is to be sent. If it is of type `SOCK_STREAM`, then this call attempts to make a connection to another socket.

Each communications space interprets the *name* parameter in its own way. This argument specifies the socket to which the socket specified in *s* is to be connected.

See also `accept`, `select`, `socket`, `getsockname`, and `shutdown` in this appendix.

### Return Values

0

Indicates success.

-1

Indicates an error; `errno` is set to one of the following:

- `EBADF` – The socket descriptor is invalid.
- `ENOTSOCK` – The socket descriptor references a file, not a socket.
- `EADDRNOTAVAIL` – The specified address is not available from the local machine.
- `EAFNOSUPPORT` – Address in the specified address family cannot be used with this socket.
- `EISCONN` – The socket is already connected.
- `ETIMEOUT` – Connection establishment timed out without establishing a connection.
- `ECONNREFUSED` – The attempt to connect was forcefully rejected.
- `ENETUNREACH` – The network is not reachable from this host.
- `EADDRINUSE` – The specified Internet address and ports are already in use.
- `EFAULT` – The *name* parameter is not a valid part of the user address space.
- `EWouldBlock` – The socket is nonblocking and the connection cannot be completed immediately. It is possible to select the socket while it is connecting by selecting it for writing.

## decc\$get\_sdc

---

## decc\$get\_sdc

Returns the Socket Device Channel (SDC) associated with a socket descriptor for direct use with the *TCP/IP Services for OpenVMS* product.

### Format

```
#include <socket.h>

short int decc$get_sdc (int s);
```

### Argument

**s**  
A socket descriptor.

### Description

The `decc$get_sdc` routine returns the SDC associated with a socket. C socket descriptors are normally used either as file descriptors or with one of the routines that take an explicit socket descriptor as its argument. C sockets are implemented using *TCP/IP Services for OpenVMS* Socket Device Channels. This routine returns the SDC used by a given socket descriptor so that you can use the *TCP/IP Services for OpenVMS*'s facilities directly by means of various I/O system services (\$QIO).

### Return Values

|   |                                                                 |
|---|-----------------------------------------------------------------|
| 0 | Indicates that <code>s</code> is not an open socket descriptor. |
| x | The SDC number.                                                 |

---

**endhostent**

Closes host database file.

**Format**

```
#include <netdb.h>

void endhostent (void);
```

**Description**

The `endhostent` routine closes the network host database file, previously opened with the `gethostbyaddr` or `gethostbyname` routine.

See also `gethostbyaddr`, and `gethostbyname` in this appendix.

**Error**

If the `endhostent` routine does not exist in your TCP/IP library, then `errno` is set to `ENOSYS`.

## endnetent

---

## endnetent

Closes the networks database file.

### Format

```
#include <netdb.h>

void endnetent (void);
```

### Description

The endnetent routine closes the networks database file, previously opened with the getnetent, setnetent, getnetbyaddr, or getnetbyname routine.

See also getnetent, getnetbyaddr, getnetbyname, and setnetent in this appendix.

### Error

If the endnetent routine does not exist in your TCP/IP library, then errno is set to ENOSYS.



---

## **endprotoent**

Closes the protocols database file.

### **Format**

```
#include <netdb.h>

void endprotoent (void);
```

### **Description**

The `endprotoent` routine closes the network protocols database file, previously opened with the `getprotoent`, `getprotobyname`, or `getprotobynumber` routine.

See also `getprotobyname`, `getprotoent`, and `getprotobynumber` in this section.

### **Error**

If the `endprotoent` routine does not exist in your TCP/IP library, then `errno` is set to `ENOSYS`.

## endservent

---

## endservent

Closes the network services database file.

### Format

```
#include <netdb.h>

void endservent (void);
```

### Description

The endservent routine closes the network services database file, previously opened with the getservent, getservbyname, or getservbyport routine.

See also getservent, getservbyname, and getservbyport in this appendix.

### Error

If the endservent routine does not exist in your TCP/IP library, then errno is set to ENOSYS.

---

## gethostaddr

Returns the standard host address for the processor.

### Format

```
#include <socket.h>

int gethostaddr (char *addr);
```

### Argument

**addr**

A pointer to the buffer in which the standard host address for the current processor is returned.

### Description

This system call returns the standard host address for the current processor. The returned address is null-terminated. The *addr* parameter must point to at least 16 bytes of free space.

Host addresses are limited to 16 characters.

### Return Values

|    |                                                                                                  |
|----|--------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                               |
| -1 | Indicates that an error has occurred and is further specified in the global <code>errno</code> . |

## gethostbyaddr

---

## gethostbyaddr

Searches the host database sequentially from the beginning of the database for a host record with a given address.

### Format

```
#include <netdb.h>

struct hostent *gethostbyaddr (char *addr, int len, int type);
```

### Routine Variants

The `gethostbyaddr` routine has a variant named `__bsd44_gethostbyaddr`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

#### **addr**

A pointer to a series of bytes in network order specifying the address of the host sought. This argument does not point to an ASCII string.

#### **len**

The number of bytes in the address pointed to by the *addr* argument.

#### **type**

The type of address format being sought. Currently, only `AF_INET` (defined in `<socket.h>`) is supported.)

### Description

The `gethostbyaddr` routine finds the first host record in the host database with the given address.

This routine uses a common static area for its return values. This means that subsequent calls this routine will overwrite any existing host entry. You must make a copy of the host entry if you wish to save it.

### Return Values

NULL

Indicates an error.

x

A pointer to an object having the `hostent` structure. See `<netdb.h>` for a description of the `hostent` structure.

Depending on the error condition and the socket implementation selected, `errno` or `h_errno` might be set to further indicate the error. (See Sections A.5 and A.6.)

---

## gethostbyname

Searches the host database sequentially from the beginning of the database for a host record with a given name or alias.

### Format

```
#include <netdb.h>

struct hostent *gethostbyname (char *name);
```

### Routine Variants

The `gethostbyname` routine has a variant named `__bsd44_gethostbyname`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Argument

#### **name**

A pointer to a null-terminated character string containing the name or an alias of the host sought.

### Description

The `gethostbyname` routine finds the first host in the host database with the given name or alias.

This routine uses a common static area for its return values. This means that subsequent calls to this routine will overwrite any existing host entry. You must make a copy of the host entry if you wish to save it.

### Return Values

NULL

Indicates an error.

x

A pointer to an object having the `hostent` structure. See `<netdb.h>` for a description of the `hostent` structure.

Depending on the error condition and the socket implementation selected, `errno` or `h_errno` might be set to further indicate the error. (See Sections A.5 and A.6.)

## gethostent

---

## gethostent

Gets a host file entry from the network host database file.

### Format

```
#include <netdb.h>

struct hostent *gethostent (void);
```

### Description

The `gethostent` routine reads the next entry of the database, opening a connection to the database, if necessary.

See the `<netdb.h>` header file for a description of the `hostent` structure.

This routine uses a common static area for its return values. Therefore, subsequent calls to this routine overwrite any existing network entry. You must make a copy of the network services entry, if you wish to save it.

### Return Values

|      |                                                  |
|------|--------------------------------------------------|
| x    | A pointer to the <code>hostent</code> structure. |
| NULL | Indicates an error.                              |

---

## gethostname

Returns the name currently associated to the host.

### Format

```
#include <socket.h>

int gethostname (char *name, int namelen); (_DECC_V4_SOURCE)
int gethostname (char *name, size_t namelen); (not _DECC_V4_SOURCE)
```

### Arguments

**name**

The address of a buffer into which the name should be written. The returned name is null-terminated unless sufficient space is not provided.

**namelen**

The size of the buffer pointed to by *name*.

### Description

The `gethostname` routine returns the hostname maintained by the *TCP/IP Services for OpenVMS* or compatible TCP/IP product.

### Return Values

|    |                                                                                                                                                                                                                                                                                                                                         |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                                                                                                                                                                                                                      |
| -1 | Indicates an error; <code>errno</code> is set to one of the following: <ul style="list-style-type: none"><li>• <code>EFAULT</code> – The buffer described by <i>name</i> and <i>namelen</i> is not a valid, writable part of the user address space.</li><li>• <code>EINVAL</code> – The returned name is an invalid address.</li></ul> |

## getnetbyaddr

---

## getnetbyaddr

Searches the network database sequentially from the beginning of the database for a network record with a given address.

### Format

```
#include <netdb.h>

struct netent *getnetbyaddr (long net, int type);
```

### Arguments

**net**

The network number, in host byte order, of the network database entry required.

**type**

The type of network sought. Currently, only AF\_INET (defined in <socket.h>) is supported.

### Description

The getnetbyaddr routine finds the first network record in the network database with the given address.

The getnetent, getnetbyaddr, and getnetbyname routines all use a common static area for their return values. This means that subsequent calls to any of these routines will overwrite any existing network entry. You must make a copy of the network entry if you wish to save it.

### Return Values

NULL

Indicates EOF or an error.

x

A pointer to an object having the netent structure. See the <netdb.h> header file for a description of the netent structure.



---

## getnetbyname

Searches the network database sequentially from the beginning of the database for a network record with a given name or alias.

### Format

```
#include <netdb.h>

struct netent *getnetbyname (char *name);
```

### Argument

**name**

A pointer to a null-terminated character string of the name or an alias of the network sought.

### Description

The getnetbyname routine finds the first host in the network database with the given name or alias.

The getnetent, getnetbyaddr, and getnetbyname routines all use a common static area for their return values. This means that subsequent calls to any of these routines will overwrite any existing network entry. You must make a copy of the network entry if you wish to save it.

### Return Values

NULL

Indicates EOF or an error; errno is set to EFAULT (The buffer described by *name* is not a valid, writable part of the user address space.)

x

A pointer to an object having the netent structure. See the <netdb.h> header file for a description of the netent structure.

Depending on the error condition and the socket implementation selected, errno or h\_errno might be set to further indicate the error. (See Sections A.5 and A.6.)

---

## getnetent

Gets a network file entry from the networks database file.

### Format

```
#include <netdb.h>

struct netent *getnetent (void);
```

### Description

The `getnetent` routine opens and sequentially reads the networks database file to retrieve network information.

See the `<netdb.h>` header file for a description of the `netent` structure.

The `getnetent` routine uses a common static area for its return values, so subsequent calls to this routine overwrite any existing network entry. You must make a copy of the network services entry, if you wish to save it.

See also `setnetent` and `endnetent` in this appendix.

### Return Values

|   |                                               |
|---|-----------------------------------------------|
| x | A pointer to a <code>netent</code> structure. |
| 0 | Indicates an error or EOF.                    |

---

## getpeername

Returns the name of the connected peer.

### Format

```
#include <socket.h>

int getpeername (int s, struct sockaddr *name, int *namelen); (_DECC_V4_SOURCE)

int getpeername (int s, struct sockaddr *name, size_t *namelen); (not _DECC_V4_SOURCE)
```

### Routine Variants

The `getpeername` routine has a variant named `__bsd44_getpeername`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

**s**

A socket descriptor that has been created using `socket`.

**name**

A pointer to a buffer within which the peer name is to be returned.

**namelen**

An address of an integer that specifies the size of the *name* buffer. On return, it will be modified to reflect the actual length, in bytes, of the *name* returned.

### Description

The `getpeername` routine returns the name of the peer connected to the socket descriptor specified.

See also `bind`, `getsockname`, and `socket` in this appendix.

### Return Values

0

Indicates success.

−1

Indicates an error; `errno` is set to one of the following:

- `EBADF` – The descriptor is invalid.
- `ENOTSOCK` – The socket descriptor references a file, not a socket.
- `ENOTCONN` – The socket is not connected.
- `ENOBUFS` – Resources were insufficient in the system to perform the operation.
- `EFAULT` – The *name* parameter is not a valid part of the user address space.

## getprotobyname

---

## getprotobyname

Searches the protocols database until a matching protocol name is found or until EOF is encountered.

### Format

```
#include <netdb.h>

struct protoent *getprotobyname (char *name);
```

### Argument

**name**

A pointer to a string containing the desired protocol name.

### Description

The getprotobyname routine returns a pointer to a protoent structure containing the broken-out fields of the requested line from the protocols database.

See the <netdb.h> header file for a description of the protoent structure.

See also getprotoent and getprotobynumber in this appendix.

### Return Values

NULL

Indicates EOF or an error.

x

A pointer to a protoent structure.

---

## getprotobynumber

Searches the protocols database until a matching protocol number is found or until an EOF is encountered.

### Format

```
#include <netdb.h>

struct protoent *getprotobynumber (int *proto);
```

### Argument

**proto**

A pointer to a string containing the desired protocol number.

### Description

The getprotobynumber routine returns a pointer to a protoent structure containing the broken-out fields of the requested line from the protocols database.

See the <netdb.h> header file for a description of the protoent structure.

See also getprotoent and getprotobynname in this appendix.

### Return Values

NULL

Indicates EOF or an error.

x

A pointer to a protoent structure.

## getprotoent

---

## getprotoent

Gets a protocol database entry from the protocols database file.

### Format

```
#include <netdb.h>

struct protoent *getprotoent (void);
```

### Description

The `getprotoent` routine reads the next entry of the database, opening a connection to the database, if necessary.

See the `<netdb.h>` header file for a description of the `protoent` structure.

See also `getprotobyname`, `getprotobynumber`, `setprotoent`, and `endprotoent` in this appendix.

### Return Values

|                   |                                                 |
|-------------------|-------------------------------------------------|
| <code>x</code>    | A pointer to a <code>protoent</code> structure. |
| <code>NULL</code> | Indicates an error or EOF.                      |

---

## getservbyname

Gets information on the named service from the network services database.

### Format

```
#include <netdb.h>

struct servent *getservbyname (char *name, char *proto);
```

### Arguments

**name**

A pointer to a string containing the name of the service about which information is required.

**proto**

A pointer to a string containing the name of the protocol to search for.

### Description

The `getservbyname` routine searches sequentially from the beginning of the file until a matching service name is found, or until an EOF is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

This routine returns a pointer to a `servent` structure containing the broken-out fields of the requested line in the network services database.

See the `<netdb.h>` header file for a description of the `servent` structure.

All information is contained in a static area, so it must be copied if it is to be saved.

See also `getservbyport` in this appendix.

### Return Values

|      |                                                |
|------|------------------------------------------------|
| NULL | Indicates EOF or an error.                     |
| x    | A pointer to a <code>servent</code> structure. |

## getservbyport

---

## getservbyport

Gets information on the specified port from the network services database.

### Format

```
#include <netdb.h>

struct servent *getservbyport (int port, char *proto);
```

### Arguments

**port**

The port number to search for.

**proto**

A pointer to a string containing the name of the protocol to search for.

### Description

The `getservbyport` routine searches sequentially from the beginning of the file until a matching port is found, or until an EOF is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

This routine returns a pointer to a `servent` structure containing the broken-out fields of the requested line in the network services database. See the `<netdb.h>` header file for a description of the `servent` structure.

All information is contained in a static area, so it must be copied if it is to be saved.

See also `getservbyname` in this appendix.

### Return Values

|      |                                                |
|------|------------------------------------------------|
| NULL | Indicates EOF or an error.                     |
| x    | A pointer to a <code>servent</code> structure. |



---

## getservent

Gets a services file entry from the network services database file.

### Format

```
#include <netdb.h>

struct servent *getservent (void);
```

### Description

The getservent routine reads the next line of the network services database file, opening a connection to the database, if necessary.

This routine returns a servent structure that contains fields for a line of information from the network services database file. See the <netdb.h> header file for a description of the hostent structure.

This routine uses a common static area for its return values, so subsequent calls to this routine overwrite any existing network entry. You must make a copy of the network services entry, if you wish to save it.

See also setservent and endservent in this appendix.

### Return Values

|      |                                   |
|------|-----------------------------------|
| x    | A pointer to a servent structure. |
| NULL | Indicates an error or EOF.        |

## getsockname

---

## getsockname

Returns the name associated with a socket.

### Format

```
#include <socket.h>
```

```
int getsockname (int s, struct sockaddr *name, int *namelen); (_DECC_V4_SOURCE)
```

```
int getsockname (int s, struct sockaddr *name, size_t *namelen); (not _DECC_V4_SOURCE)
```

### Routine Variants

The `getsockname` routine has a variant named `__bsd44_getsockname`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

#### **s**

A socket descriptor created with `socket` and bound to the socket name with `bind`.

#### **name**

A pointer to the buffer in which `getsockname` should return the socket name.

#### **namelen**

A pointer to an integer specifying the size of the buffer pointed to by *name*. On return, the integer contains the actual size of the name returned, in bytes.

### Description

The `getsockname` routine returns the current name for the specified socket descriptor. The name is a format specific to the address family (`AF_INET`) assigned to the socket.

`bind` makes the association of the name to the socket, not `getsockname`.

See also `bind` and `socket` in this appendix.

### Return Values

0

Indicates success.

−1

Indicates an error; `errno` is set to one of the following:

- `EBADF` – The descriptor is invalid.
- `ENOTSOCK` – The socket descriptor references a file, not a socket.
- `ENOBUFS` – Resources were insufficient in the system to perform the operation.
- `EFAULT` – The *name* parameter is not a valid part of the user address space.

---

## getsockopt

Returns the options set on a socket.

### Format

```
#include <socket.h>
```

```
int getsockopt (int s, int level, int optname, char *optval, int *optlen); (_DECC_V4_SOURCE)
```

```
int getsockopt (int s, int level, int optname, void *optval, size_t *optlen); (not _DECC_V4_SOURCE)
```

### Arguments

#### **s**

A socket descriptor created by socket.

#### **level**

The protocol level for which the socket options are desired. It may have one of the following values:

SOL\_SOCKET

Get the options at the socket level.

p

Any protocol number. Get the options for protocol level p. See the <in.h> file for the various IPPROTO values.

#### **optname**

Is interpreted by the protocol that is specified in the level. Options at each protocol level are documented with the protocol. See setsockopt in this appendix for socket level options.

#### **optval**

Points to a buffer in which the value of the specified option should be placed by getsockopt.

#### **optlen**

Points to an integer containing the size of the buffer pointed to by *optval*. On return, the integer will be modified to contain the actual size of the option value returned.

### Description

The getsockopt routine gets information on socket options. See the appropriate protocol for information on available options at each protocol level.

### Return Values

0

Indicates success.

## getsockopt

–1

Indicates an error; `errno` is set to one of the following:

- `EBADF` – The descriptor is invalid.
- `ENOTSOCK` – The socket descriptor references a file, not a socket.
- `ENOPROTOOPT` – The option is unknown or the protocol is unsupported.

---

**herror**

Writes a message to standard error explaining `h_error`.

**Format**

```
#include <netdb.h>
void herror (const char *string);
```

**Argument**

**string**  
A user-printable string.

**Description**

The `herror` routine maps the error number in the external variable `h_errno` to a locale-dependent error message.

---

## hstrerror

Returns an error message string.

### Format

```
#include <string.h>

char *hstrerror (int errnum);
```

### Arguments

**errnum**  
An error number specifying a value of `h_errno`.

### Description

The `hstrerror` routine maps the error number specified by the *errnum* parameter to a locale-dependent error message string and returns a pointer to the string. The string pointed to by the return value cannot be modified by the program, but could be overwritten by subsequent calls to this routine.

### Return Values

|                 |                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------|
| <code>x</code>  | A pointer to the generated message string.                                                       |
| <code>-1</code> | On error, <code>errno</code> might be set, but no return value is reserved to indicate an error. |

### Errors

If the `hstrerror` routine fails, `errno` is set to one of the following values:

- `EINVAL` – The *errnum* parameter is an invalid error number.

---

## hostalias

Searches for host aliases associated with a name.

### Format

```
#include <resolv.h>

char *hostalias (const char *name);
```

### Argument

**name**

Points to the name of the host that you want to retrieve aliases from.

### Description

The `hostalias` routine searches for the alias associated with the *name* parameter. The `HOSTALIASES` environment variable defines the name of a file where you can find the host aliases, in the form:

```
host alias
```

### Return Values

|      |                     |
|------|---------------------|
| x    | The host alias.     |
| NULL | Indicates an error. |

## htonl

---

## htonl

Converts longwords from host to network byte order.

### Format

```
#include <in.h>

unsigned long int htonl (unsigned long int hostlong);
```

### Argument

#### **hostlong**

A longword in host (VAX system) byte order. All integers on VAX systems are in host byte order unless otherwise specified.

### Description

The `htonl` routine converts 32-bit unsigned integers from host byte order to network byte order.

The network byte order is the format in which data bytes are supposed to be transmitted through a network. All hosts on a network must send data in network byte order. Not all hosts have an internal data representation format that is identical to the network byte order. The host byte order is the format in which bytes are ordered internally on a specific host.

The host byte order on VAX systems differs from the network order.

This routine is most often used with Internet addresses and ports as returned by `gethostent` and `getservent`, and when manipulating values in the structures. Network byte-order places the byte with the most significant bits at lower addresses; VAX systems place the most significant bits at the highest address.

### Return Value

|   |                                   |
|---|-----------------------------------|
| x | A longword in network byte order. |
|---|-----------------------------------|



---

## htons

Converts short integers from host to network byte order.

### Format

```
#include <in.h>
```

```
unsigned short int htons (unsigned short int hostshort);
```

### Argument

#### **hostshort**

A short integer in host (VAX system) byte order. All short integers on VAX systems are in host byte order unless otherwise specified.

### Description

The `htons` routine converts 16-bit unsigned integers from host byte order to network byte order.

The network byte order is the format in which data bytes are supposed to be transmitted through a network. All hosts on a network must send data in network byte order. Not all hosts have an internal data representation format that is identical to the network byte order. The host byte order is the format in which bytes are ordered internally on a specific host.

The host byte order on VAX systems differs from the network order.

This routine is most often used with Internet addresses and ports as returned by `gethostent` and `getservent`, and when manipulating values in the structures. Network byte order places the byte with the most significant bits at lower addresses; VAX systems place the most significant bits at the highest address.

### Return Value

x

A short integer in network byte order. Integers in network byte order cannot be used for arithmetic computation on VAX systems.

---

## inet\_addr

Converts Internet addresses in text form into numeric (binary) Internet addresses.

### Format

```
#include <in.h>
#include <inet.h>
int inet_addr (char *cp);
```

### Argument

#### **cp**

A pointer to a null-terminated character string containing an Internet address in the standard Internet "." format.

### Description

The `inet_addr` routine returns an Internet address in network byte order when given as its argument an ASCIZ (null-terminated) string representing the address in the Internet standard "." notation.

Internet addresses specified using the "." notation take one of the following forms:

```
a.b.c.d
a.b.c
a.b
a
```

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address. Note that when an Internet address is viewed as a 32-bit integer quantity on VAX systems, the bytes previously referred to appear in binary as "d.c.b.a". That is, VAX bytes are ordered from least significant to most significant.

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as "parts" in a "." address expression may be decimal, octal, or hexadecimal, as specified in the C language (that is, a leading 0x or 0X implies hexadecimal; a leading 0 implies octal; otherwise, the number is interpreted as decimal).

### Return Values

|    |                                                                       |
|----|-----------------------------------------------------------------------|
| -1 | Indicates that <i>cp</i> does not point to a proper Internet address. |
| x  | An Internet address in network byte order.                            |

---

## inet\_lnaof

Returns the local network address portion of an Internet address.

### Format

```
#include <in.h>
#include <inet.h>
int inet_lnaof (struct in_addr in);
```

### Argument

**in**  
An Internet address.

### Description

The `inet_lnaof` routine returns the local network address (LNA) portion of a full Internet address.

### Return Value

|   |                                                            |
|---|------------------------------------------------------------|
| x | The LNA portion of an Internet address in host byte order. |
|---|------------------------------------------------------------|

## inet\_makeaddr

---

## inet\_makeaddr

Returns an Internet address given a network address and a local address on that network.

### Format

```
#include <in.h>
#include <inet.h>
struct in_addr inet_makeaddr (int net, int lna);
```

### Arguments

**net**

An Internet network address in host byte order.

**lna**

A local network address on network *net* in host byte order.

### Description

The `inet_makeaddr` routine combines the *net* and *lna* arguments into a single Internet address.

### Return Value

|   |                                            |
|---|--------------------------------------------|
| x | An Internet address in network byte order. |
|---|--------------------------------------------|

---

## inet\_netof

Returns the Internet network address portion of an Internet address.

### Format

```
#include <in.h>
#include <inet.h>
int inet_netof (struct in_addr in);
```

### Argument

**in**  
An Internet address.

### Description

The `inet_netof` routine returns the Internet network address (NET) portion of a full Internet address.

### Return Value

|   |                                                                         |
|---|-------------------------------------------------------------------------|
| x | The Internet network portion of an Internet address in host byte order. |
|---|-------------------------------------------------------------------------|

## inet\_network

---

## inet\_network

Converts a text string representing an Internet network address in the standard Internet "." notation into an Internet network address as machine-format integer values.

### Format

```
#include <in.h>
#include <inet.h>

int inet_network (char *cp);
```

### Argument

#### **cp**

A pointer to an ASCIZ (null-terminated) character string containing a network address in the standard Internet "." format.

### Description

The `inet_network` routine returns an Internet network address as machine-format integer values when given as its argument an ASCIZ string representing the address in the Internet standard "." notation.

### Return Values

|    |                                                                               |
|----|-------------------------------------------------------------------------------|
| -1 | Indicates that <i>cp</i> does not point to a proper Internet network address. |
| x  | An Internet network address as machine-format integer values.                 |

---

## inet\_ntoa

Converts an Internet address into a text string representing the address in the standard Internet "." notation.

### Format

```
#include <in.h>
#include <inet.h>
char *inet_ntoa (struct in_addr in);
```

### Argument

**in**

An Internet address in network byte order.

### Description

The `inet_ntoa` routine converts an Internet address into an ASCIIZ (null-terminated) string representing that address in the standard Internet "." notation.

---

#### Warning

---

Arguments should not be passed as integers because of how *Compaq C* handles struct arguments.

---

Because the string is returned in a static buffer that will be overwritten by successive calls to `inet_ntoa`, it is recommended to copy the string to a safe place.

### Return Value

x

A pointer to a string containing the Internet address in "." notation.

## ioctl

---

## ioctl

Controls socket operations only.

### Format

```
#include <ioctl.h>

int ioctl (int d, unsigned long request, void *arg);
```

### Arguments

#### **d**

Specifies the file descriptor of the requested device.

#### **request**

Specifies the ioctl command performed on the device.

#### **arg**

Specifies parameters for this request. The type of *arg* is dependent on the specific ioctl request and device to which the ioctl is targeted.

### Description

The ioctl routine performs a variety of operations on sockets.

An ioctl request has encoded in it, whether the parameter is an "in" parameter or "out" parameter, and the size of the *arg* parameter in bytes. The ioctl request is defined in the <ioctl.h> header file.

### Return Values

|    |                                                            |
|----|------------------------------------------------------------|
| 0  | Indicates success.                                         |
| -1 | Indicates an error; further specified in the global errno. |

### Errors

If the ioctl routine fails, errno is set to one of the following values:

- EBADF – The *d* parameter is not a valid descriptor.
- ENOTTY – The *d* parameter is not associated with a character special device, or the specified request does not apply to the kind of object that the *d* parameter references.
- EINVAL – Either the *request* or the *arg* parameter is not valid.



---

## listen

Sets the maximum limit of outstanding connection requests for a socket that is connection-oriented.

### Format

```
int listen (int s, int backlog);
```

### Arguments

**s**

A socket descriptor of type SOCK\_STREAM that has been created using socket.

**backlog**

The maximum number of pending connections that may be queued on the socket at any given time. The maximum cannot exceed 5.

### Description

The listen routine creates a queue for pending connection requests on socket *s* with a maximum size of *backlog*. Connections may then be accepted with accept.

If a connection request arrives with the queue full (more than *backlog* connection requests pending), the client will receive a timeout.

See also accept, connect, and socket in this appendix.

### Return Values

0

Indicates success.

–1

Indicates an error; errno is set to one of the following values:

- EBADF – The socket descriptor is invalid.
- ENOTSOCK – The socket descriptor references a file, not a socket.
- EOPNOTSUPP – The socket is not of a type that supports the operation listen.

## ntohl

---

## ntohl

Converts longwords from network to host byte order.

### Format

```
#include <in.h>

unsigned long int ntohl (unsigned long int netlong);
```

### Argument

#### **netlong**

A longword in network byte order. Integers in network byte order cannot be used for arithmetic computation on VAX systems.

### Description

The `ntohl` routine converts 32-bit unsigned integers from network byte order to host byte order.

The network byte order is the format in which data bytes are supposed to be transmitted through a network. All hosts on a network must send data in network byte order. Not all hosts have an internal data representation format that is identical to the network byte order. The host byte order is the format in which bytes are ordered internally on a specific host.

The host byte order on VAX systems differs from the network order.

This routine is most often used with Internet addresses and ports as returned by `gethostent` and `getservent`, and when manipulating values in the structures. Network byte order places the byte with the most significant bits at lower addresses; VAX systems place the most significant bits at the highest address.

### Return Value

|   |                                |
|---|--------------------------------|
| x | A longword in host byte order. |
|---|--------------------------------|

---

## ntohs

Converts short integers from network to host byte order.

### Format

```
#include <in.h>

unsigned short int ntohs (unsigned short int netshort);
```

### Argument

#### **netshort**

A short integer in network byte order. Integers in network byte order cannot be used for arithmetic computation on VAX systems.

### Description

The ntohs routine converts 16-bit unsigned integers from network byte order to host byte order.

The network byte order is the format in which data bytes are supposed to be transmitted through a network. All hosts on a network must send data in network byte order. Not all hosts have an internal data representation format that is identical to the network byte order. The host byte order is the format in which bytes are ordered internally on a specific host.

The host byte order on VAX systems differs from the network order.

This routine is most often used with Internet addresses and ports as returned by `gethostent` and `getservent`, and when manipulating values in the structures. Network byte order places the byte with the most significant bits at lower addresses; VAX systems place the most significant bits at the highest address.

### Return Value

|   |                                           |
|---|-------------------------------------------|
| x | A short integer in host (VAX) byte order. |
|---|-------------------------------------------|

## read

---

## read

Reads bytes from a socket or file and places them in a buffer.

### Format

```
#include <unistd.h>

int read (int d, void *buffer, int nbytes);
```

### Arguments

#### **d**

A descriptor that must refer to a socket or file currently opened for reading.

#### **buffer**

The address of contiguous storage in which the input data is placed.

#### **nbytes**

The maximum number of bytes involved in the read operation.

### Description

If the end-of-file is not reached, the read routine returns *nbytes*. If the end-of-file occurs during the read routine, it returns the number of bytes read.

Upon successful completion, read returns the number of bytes actually read and placed in the buffer.

See also socket in this appendix.

### Return Values

x

The number of bytes read and placed in the file.

– 1

Indicates an error; *errno* is set to one of the following:

- EBADF – The socket descriptor is invalid.
- EFAULT – The *buffer* points outside the allocated address space.
- EINVAL – The *nbytes* argument is negative.
- EWOULDBLOCK – The *NBIO* socket option (nonblocking) flag is set for the socket or file descriptor and the process would be delayed in the read operation.

---

## recv

Receives bytes from a connected socket and places them into a buffer.

### Format

```
#include <socket.h>

int recv (int s, char *buf, int len, int flags); (_DECC_V4_SOURCE)

ssize_t recv (int s, void *buf, size_t len, int flags); (not _DECC_V4_SOURCE)
```

### Arguments

#### **s**

A socket descriptor that was created as the result of a call to accept or connect.

#### **buf**

A pointer to a buffer into which received data will be placed.

#### **len**

The size of the buffer pointed to by *buf*.

#### **flags**

A bit mask that may contain one or more of MSG\_OOB and MSG\_PEEK. It is built by ORing the appropriate values together.

The MSG\_OOB flag allows out-of-band data to be received. If out-of-band data is available, it will be read before any other data that is available. If no out-of-band data is available, the MSG\_OOB flag is ignored. Out-of-band data can be sent using send, sendmsg, and sendto.

The MSG\_PEEK flag allows you to peek at the data that is next in line to be received without removing it from the system's buffers.

### Description

The recv routine receives data from a connected socket. To receive data on an unconnected socket, use the recvfrom or recvmsg routines. The received data is placed in the buffer *buf*.

Data is sent by the socket's peer using the send, sendmsg, or sendto routines.

You may use the select routine to determine when more data arrives.

If no data is available at the socket, the receive call waits for data to arrive, unless the socket is nonblocking in which case a -1 is returned with the external variable errno set to EWOULDBLOCK.

See also read, send, sendmsg, sendto, and socket in this appendix.

### Return Values

|   |                                                             |
|---|-------------------------------------------------------------|
| x | The number of bytes received and placed in <i>buf</i> .     |
| 0 | Indicates that a connection is broken or reset by its peer. |

–1

Indicates an error; `errno` is set to one of the following:

- `EBADF` – The socket descriptor is invalid.
- `ENOTSOCK` – The descriptor references a file, not a socket.
- `EPIPE` – An attempt was made to write to a socket that is not open for reading by any process.
- `EWouldBlock` – The `NBIO` socket option (nonblocking) flag is set for the socket or file descriptor and the process would be delayed in the read operation.
- `EFAULT` – The data was specified to be received into a non-existent or protected part of the process address space.

---

## recvfrom

Receives bytes from a socket from any source.

### Format

```
#include <socket.h>

int recvfrom (int s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen) ;
 (_DECC_V4_SOURCE)

ssize_t recvfrom (int s, void *buf, size_t len, int flags, struct sockaddr *from, size_t *fromlen) ;
 (not _DECC_V4_SOURCE)
```

### Routine Variants

The `recvfrom` routine has a variant named `__bsd44_recvfrom`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

#### **s**

A socket descriptor that has been created with `socket` and bound to a name using `bind` or as a result of `accept`.

#### **buf**

A pointer to a buffer into which received data will be placed.

#### **len**

The size of the buffer pointed to by *buf*.

#### **flags**

A bit mask that may contain one or more of `MSG_OOB` and `MSG_PEEK`. It is built by ORing the appropriate values together.

The `MSG_OOB` flag allows out-of-band data to be received. If out-of-band data is available, it will be read before any other data that is available. If no out-of-band data is available, the `MSG_OOB` flag is ignored. Out-of-band data can be sent using `send`, `sendmsg`, and `sendto`.

The `MSG_PEEK` flag allows you to peek at the data that is next in line to be received without actually removing it from the system's buffers.

#### **from**

If *from* is nonzero, *from* is a buffer into which `recvfrom` places the address (structure) of the socket from which the data is received. If *from* was 0, the address will not be returned.

#### **fromlen**

Points to an integer containing the size of the buffer pointed to by *from*. On return, the integer is modified to contain the actual length of the socket address structure returned.

## recvfrom

### Description

The `recvfrom` routine allows a named, unconnected socket to receive data. The data is placed in the buffer pointed to by *buf*, and the address of the sender of the data is placed in the buffer pointed to by *from* if *from* is non-NULL. The structure that *from* points to is assumed to be as large as the `sockaddr` structure. See `<socket.h>` for a description of the `sockaddr` structure.

To receive bytes from any source, the sockets need not be connected to another socket.

You may use the `select` routine to determine if data is available.

If no data is available at the socket, the `recv` call waits for data to arrive, unless the socket is nonblocking, in which case a `-1` is returned with the external variable `errno` set to `EWouldBlock`.

See also `read`, `send`, `sendmsg`, `sendto`, and `socket` in this appendix.

### Return Values

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x</code>  | The number of bytes of data received and placed in <i>buf</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <code>-1</code> | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"><li>• <code>EBADF</code> – The socket descriptor is invalid.</li><li>• <code>ENOTSOCK</code> – The socket descriptor references a file, not a socket.</li><li>• <code>EPIPE</code> – An attempt was made to write to a socket that is not open for reading by any process.</li><li>• <code>EWouldBlock</code> – The <code>NBIO</code> (nonblocking) flag is set for the socket descriptor and the process would be delayed in the write operation.</li><li>• <code>EFAULT</code> – The data was specified to be received into a non-existent or protected part of the process address space.</li></ul> |



---

## recvmsg

Receives bytes on a socket and places them into scattered buffers..

### Format

```
#include <socket.h>

int recvmsg (int s, struct msghdr msg[], int flags);
```

### Routine Variants

The `recvmsg` routine has a variant named `__bsd44_recvmsg`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

#### **s**

A socket descriptor that has been created with `socket`.

#### **msg**

A `msghdr` structure. See `<socket.h>` for a description of the `msghdr` structure.

#### **flags**

A bit mask that may contain one or more of `MSG_OOB` and `MSG_PEEK`. It is built by ORing the appropriate values together.

The `MSG_OOB` flag allows out-of-band data to be received. If out-of-band data is available, it will be read before any normal data that is available. If no out-of-band data is available, the `MSG_OOB` flag is ignored. Out-of-band data can be sent using `send`, `sendmsg`, and `sendto`.

The `MSG_PEEK` flag allows you to peek at the data that is next in line to be received without actually removing it from the system's buffers.

### Description

The `recvmsg` routine may be used with any socket, whether it is in a connected state or not. It receives data sent by a call to `sendmsg`, `send`, or `sendto`. The message is scattered into several user buffers if such buffers are specified.

To receive data, the socket need not be connected to another socket.

When the `iovec[iovcnt]` array specifies more than one buffer, the input data is scattered into `iovcnt` buffers as specified by the members of the `iovec` array:

```
iov[0], iov[1], ..., iov[iovcnt]
```

When a message is received, it is split among the buffers by filling the first buffer in the list, then the second, and so on, until either all of the buffers are full or there is no more data to be placed in the buffers.

When a message is sent, the first buffer is copied to a system buffer and then the second buffer is copied, followed by the third buffer and so on, until all the buffers are copied. After the data is copied, the protocol will send the data to the remote host at the appropriate time, depending upon the protocol.

You may use the `select` routine to determine when more data arrives.

## recvmsg

See also `read`, `send`, and `socket` in this appendix.

### Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | The number of bytes returned in the <code>msg_iov</code> buffers.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| -1 | <p>Indicates an error; <code>errno</code> is set to one of the following values:</p> <ul style="list-style-type: none"><li>• <code>EBADF</code> – The socket descriptor is invalid.</li><li>• <code>ENOTSOCK</code> – The socket descriptor references a file, not a socket.</li><li>• <code>EPIPE</code> – An attempt was made to write to a socket that is not open for reading by any process.</li><li>• <code>EWouldBlock</code> – The <code>NBIO</code> (nonblocking) flag is set for the socket descriptor and the process would be delayed in the write operation.</li><li>• <code>EINTR</code> – The receive was interrupted by delivery of a signal before any data was available for the receive.</li><li>• <code>EFAULT</code> – The data was specified to be received into a non-existent or protected part of the process address space.</li></ul> |

---

## select

Allows the user to poll or check a group of sockets for I/O activity. It can check which sockets are ready to be read or written, or which sockets have a pending exception.

As of OpenVMS Version 7.0, the select routine can operate on any number of sockets up to the limit of open files supported by the *Compaq C* RTL (65535 on OpenVMS Alpha; 2047 on OpenVMS VAX). However, for select to use more than 32 sockets, the *TCP/IP Services for OpenVMS* used with the *Compaq C* RTL must offer the same support.

## Format

```
#include <socket.h>

int select (int nfds, int *readfds, int *writefds, int *exceptfds, struct timeval *timeout);
 (_DECC_V4_SOURCE)

int select (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
 (not _DECC_V4_SOURCE)
```

## Arguments

### **nfds**

The highest numbered socket descriptor to search for. That is, the highest numbered bit + 1 in *readfds*, *writefds*, and *exceptfds* that should be examined. Descriptor *s* is represented by  $1 \ll s$  (1 shifted to the left *s* number of times).

---

### **Note**

---

You must increase the value of FD\_SETSIZE if your application will use more sockets in the call to select than the value of FD\_SETSIZE in the `<time.h>` or `<socket.h>` header files.

Do this by defining a larger value for FD\_SETSIZE within the application before including `<time.h>` or `<socket.h>`.

---

### **readfds**

A pointer to an array of bits, organized as integers (each integer describing 32 descriptors), that should be examined for read readiness. If bit *n* of the longword is set, socket descriptor *n* will be checked to see if it is ready to be read. All bits set in the bit mask must correspond to the file descriptors of sockets. The select routine cannot be used on normal files.

On return, the array of bits to which *readfds* points contains a bit mask of the sockets that are ready for reading. Only bits that are set on entry to select will be set on exit.

If this pointer is NULL, select ignores the array.

### **writefds**

A pointer to a longword bit mask of the socket descriptors that should be examined for write readiness. If bit *n* of the longword is set, socket descriptor *n* will be checked to see if it is ready to be written to. All bits set in the bit mask must correspond to socket descriptors.

## select

On return, the array of bits that *writelfds* points to contains a bit mask of the sockets that are ready for writing. Only bits that are set on entry to select will be set on exit.

If this pointer is NULL, select ignores the array.

### exceptfds

A pointer to a longword bit mask of the socket descriptors that should be examined for exceptions. If bit *n* of the longword is set, socket descriptor *n* will be checked to see if it has any pending exceptions. All bits set in the bit mask must correspond to the file descriptors of sockets.

On return, the array of bits pointer *exceptfds* contains a bit mask of the sockets that have exceptions pending. Only bits that are set on entry to select will be set on exit.

If this pointer is NULL, select ignores the array.

### timeout

The length of time that select should examine the sockets before returning. If one of the sockets specified in the *readfds*, *writelfds*, and *exceptfds* bit masks is ready for I/O, select will return before the timeout period has expired.

The *timeout* structure points to a timeval structure. See <socket.h> for a description of the timeval structure.

## Description

The select routine determines the I/O status of the sockets specified in the various mask arguments. It returns either when a socket is ready to be read or written, or when the *timeout* period expires. If *timeout* is a nonzero integer, it specifies a maximum interval to wait for the selection to complete.

If the *timeout* argument is NULL, select will block indefinitely. In order to effect a poll, *timeout* should be non-NULL, and should point to a zero-valued structure.

If a process is blocked on a select while waiting for input from a socket and the sending process closes the socket, the select notes this as an event and will unblock the process. The descriptors are always modified on return if select returns because of the *timeout*.

When the socket option SO\_OOBINLINE is set on the device\_socket, a select on both read and exception events returns the socket mask set on both the read and exception mask. Otherwise, only the exception mask is set.

---

### Note

---

Beginning with OpenVMS Version 7.2-1, the select function fails if one of the specified file descriptor sets contains an invalid file descriptor (errno is set to EBADF) or a file descriptor not associated with a socket (errno is set to ENOTSOCK).

Previously, select set errno as described, but was otherwise ignoring invalid file descriptors and file descriptors not associated with sockets. To request this old behavior, define the logical name DECC\$SELECT\_IGNORES\_INVALID\_FD to any value before invoking the application.

---

See also accept, connect, read, recv, recvfrom, recvmsg, send, sendmsg, sendto, and write in this appendix.

**Return Values**

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| n  | The number of sockets that were ready for I/O or that had pending exceptions. This value matches the number of returned bits that are set in all output masks.                                                                                                                                                                                                                                                                                              |
| 0  | Indicates that <code>select</code> timed out before any socket became ready for I/O.                                                                                                                                                                                                                                                                                                                                                                        |
| -1 | Indicates an error; <code>errno</code> is set to one of the following values: <ul style="list-style-type: none"><li>• <b>EBADF</b> – One of the bit masks specified an invalid descriptor.</li><li>• <b>EINVAL</b> – The specified time limit is unacceptable. One of its components is negative or too large.</li><li>• <b>ENOTSOCK</b> – A file descriptor not associated with a socket was found in one of the specified file descriptor sets.</li></ul> |

## send

---

## send

Sends bytes through a socket to its connected peer.

### Format

```
#include <socket.h>

int send (int s, char *msg, int len, int flags); (_DECC_V4_SOURCE)

ssize_t send (int s, const void *msg, size_t len, int flags); (not _DECC_V4_SOURCE)
```

### Arguments

#### **s**

A socket descriptor that was created with `socket`, and connected to another socket using `accept` or `connect`.

#### **msg**

A pointer to a buffer containing the data to be sent.

#### **len**

The length, in bytes, of the data pointed to by *msg*.

#### **flags**

May be either 0 or `MSG_OOB`. If it is equal to `MSG_OOB`, the data will be sent out-of-band. This means that the data can be received before other pending data on the receiving socket if the receiver also specifies a `MSG_OOB` in the flag parameter of the call.

### Description

The `send` routine may only be used on connected sockets. To send data on an unconnected socket, use the `sendmsg` or `sendto` routines. The `send` routine passes data along to its connected peer, which may receive the data by using `recv`.

If there is no space available to buffer the data being sent on the receiving end of the connection, `send` will normally block until buffer space becomes available. If the socket is defined as nonblocking, however, `send` will fail with an `errno` indication of `EWOULDBLOCK`. If the message is too large to be sent in one piece and the socket type requires that messages be sent atomically (using `SOCK_DGRAM`), `send` will fail with an `errno` indication of `EMSGSIZE`.

No indication of failure to deliver is implicit in a `send`. All errors (except `EWOULDBLOCK`) are detected locally. You may use the `select` routine to determine when it is possible to send more data.

See also `read`, `recv`, `recvmsg`, `recvfrom`, `getsockopt`, and `socket` in this appendix.

## Return Values

n

The number of bytes sent. This value will normally equal *len*.

−1

Indicates an error; *errno* is set to one of the following:

- EBADF – The socket descriptor is invalid.
- ENOTSOCK – The descriptor references a file, not a socket.
- EFAULT – An invalid user space address was specified for a parameter.
- EMSGSIZE – The socket requires that messages be sent atomically, and the size of the message to be sent made this impossible.
- EWOULDBLOCK – Blocks if the system does not have enough space for buffering the user data.

## sendmsg

---

## sendmsg

Sends gathered bytes through a socket to any other socket.

### Format

```
#include <socket.h>

int sendmsg (int s, struct msghdr msg[], int flags);
```

### Routine Variants

The `sendmsg` routine has a variant named `__bsd44_sendmsg`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See Section A.7 for more information.

### Arguments

#### **s**

A socket descriptor that has been created with `socket`.

#### **msg**

A pointer to a `msghdr` structure containing the message to be sent. See `<socket.h>` for a description of the `msghdr` structure.

The `msg_iov` field of the `msghdr` structure is used as a series of buffers from which data is read in order until `msg_iovlen` bytes have been obtained.

#### **flags**

May be either 0 or `MSG_OOB`. If it is equal to `MSG_OOB`, the data will be sent out-of-band. This means that the data can be received before other pending data on the receiving socket if the receiver also specifies a flag of `MSG_OOB`.

### Description

The `sendmsg` routine may be used on any socket to send data to any named socket. The data in the `msg_iovec` field of the `msg` structure is sent to the socket whose address is specified in the `msg_name` field of the structure. The receiving socket gets the data using either the `read`, `recv`, `recvfrom`, or `recvmsg` routine. When the `iovec` array specifies more than one buffer, the data is gathered from all specified buffers before being sent. See `<uio.h>` for a description of the `iovec` structure.

If there is no space available to buffer the data being sent on the receiving end of the connection, `sendmsg` will normally block until buffer space becomes available. If the socket is defined as nonblocking, however, `sendmsg` will fail with an `errno` indication of `EWOULDBLOCK`. If the message is too large to be sent in one piece and the socket type requires that messages be sent atomically (`SOCK_DGRAM`), `sendmsg` will fail with an `errno` indication of `EMSGSIZE`.

If the address specified is a `INADDR_BROADCAST` address, then the `SO_BROADCAST` option must be set and the process must have `SYSPRV` or `BYPASS` privilege for the I/O operation to succeed.

No indication of failure to deliver is implicit in a `sendmsg`. All errors (except `EWOULDBLOCK`) are detected locally. You may use the `select` routine to determine when it is possible to send more data.



See also `read`, `recv`, `recvfrom`, `recvmsg`, `socket`, and `getsockopt` in this appendix.

## Return Values

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>n</code>  | The number of bytes sent.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>-1</code> | <p>Indicates an error; <code>errno</code> is set to one of the following:</p> <ul style="list-style-type: none"><li>• <code>EBADF</code> – The socket descriptor is invalid.</li><li>• <code>ENOTSOCK</code> – The descriptor references a file, not a socket.</li><li>• <code>EFAULT</code> – An invalid user space address was specified for a parameter.</li><li>• <code>EMSGSIZE</code> – The socket requires that messages be sent atomically, and the size of the message to be sent made this impossible.</li><li>• <code>EWOULDBLOCK</code> – Blocks if the system does not have enough space for buffering the user data.</li></ul> |

## sendto

---

## sendto

Sends bytes through a socket to any other socket.

### Format

```
#include <socket.h>

int sendto (int s, char *msg, int len, int flags, struct sockaddr *to, int tolen); (_DECC_V4_SOURCE)

ssize_t sendto (int s, const void *msg, size_t len, int flags, const struct sockaddr *to, size_t tolen);
(not _DECC_V4_SOURCE)
```

### Routine Variants

The `sendto` routine has a variant named `__bsd44_sendto`. Enabled by defining `_SOCKADDR_LEN`, this variant implements 4.4BSD-compatible semantics. See [Section A.7](#) for more information.

### Arguments

#### **s**

A socket descriptor that has been created with `socket`.

#### **msg**

A pointer to a buffer containing the data to be sent.

#### **len**

The length of the data pointed to by *msg*.

#### **flags**

May be either 0 or `MSG_OOB`. If it is equal to `MSG_OOB`, the data will be sent out-of-band. This means that the data can be received before other pending data on the receiving socket if the receiver also specifies a `MSG_OOB` in its flag parameter of the call.

#### **to**

Points to the address structure of the socket to which the data is to be sent.

#### **tolen**

The length of the address structure *to* points to.

### Description

The `sendto` routine may be used on any socket to send data to any named socket. The data in the *msg* buffer is sent to the socket whose address is specified in *to*, and the address of socket *s* is provided to the receiving socket. The receiving socket gets the data using either the `read`, `recv`, `recvfrom`, or `recvmsg` routine.

If there is no space available to buffer the data being sent on the receiving end of the connection, `sendto` will normally block until buffer space becomes available. If the socket is defined as nonblocking, however, `sendto` will fail with an `errno` indication of `EWOULDBLOCK`. If the message is too large to be sent in one piece and the socket type requires that messages be sent atomically (using `SOCK_DGRAM`), `sendto` will fail with an `errno` indication of `EMSGSIZE`.

No indication of failure to deliver is implicit in a `sendto`. All errors (except `EWouldBlock`) are detected locally. You may use the `select` routine to determine when it is possible to send more data.

If the address specified is a `INADDR_BROADCAST` address, then the `SO_BROADCAST` option must be set and the process must have `SYSPRV` or `BYPASS` privilege for the I/O operation to succeed.

See also `read`, `recv`, `recvfrom`, `recvmsg`, `socket`, and `getsockopt` in this appendix.

## Return Values

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>n</code>  | The number of bytes sent. This value will normally equal <i>len</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <code>-1</code> | Indicates an error; <code>errno</code> is set to one of the following: <ul style="list-style-type: none"> <li>• <code>EBADF</code> – The socket descriptor is invalid.</li> <li>• <code>ENOTSOCK</code> – The descriptor references a file, not a socket.</li> <li>• <code>EFAULT</code> – An invalid user space address was specified for a parameter.</li> <li>• <code>EMSGSIZE</code> – The socket requires that messages be sent atomically, and the size of the message to be sent made this impossible.</li> <li>• <code>EWouldBlock</code> – Blocks if the system does not have enough space for buffering the user data.</li> </ul> |

## sethostent

---

## sethostent

Opens and rewinds the network host database file.

### Format

```
#include <netdb.h>

void sethostent (int stay_open);
```

### Argument

#### **stay\_open**

Specifies a value used to indicate when to close the host database file:

- 0 (zero) to close the host database file after each call to the `gethostbyname` or `gethostbyaddr` routine.
- Nonzero to keep the host database file open after each call.

### Description

The `sethostent` routine opens or rewinds the network host database file and resets the file marker to the beginning of the file.

Passing a nonzero value to the *stay\_open* parameter keeps the connection open until the `endhostent` or `exit` routine is called.

See also `endhostent` in this appendix.

---

## setnetent

Opens and rewinds the networks database file.

### Format

```
#include <netdb.h>

void setnetent (int stay_open);
```

### Argument

#### **stay\_open**

Specifies a value used to indicate when to close the networks database file:

- 0 (zero) to close the networks database file after each call to the `getnetent`.
- Nonzero to keep the networks database file open after each call.

### Description

The `setnetent` routine opens the networks database file and resets the file marker to the beginning of the file.

Passing a nonzero value to the *stay\_open* parameter keeps the connection open until you call the `endnetent` or `exit` routine.

See also `endnetent`, `getnetent`, and `exit` in this appendix.

---

## setprotoent

Opens, rewinds, or closes the protocols database file.

### Format

```
#include <netdb.h>

void setprotoent (int stay_open);
```

### Argument

#### **stay\_open**

Specifies a value used to indicate when to close the protocols database file:

- 0 (zero) to close the host database file after each call to the `getprotoent` routine.
- Nonzero to keep the host database file open after each call.

### Description

The `setprotoent` routine opens or rewinds the protocols database file and sets the file marker to the beginning of the file.

Passing a nonzero value to the `stay_open` parameter keeps the connection open until you call the `endprotoent`, or `exit` routine.

See also `endprotoent`, `exit`, and `getprotoent` in this appendix.

### Return Values

|   |                                                                 |
|---|-----------------------------------------------------------------|
| 1 | Indicates success.                                              |
| 0 | Indicates an error; unable to open the protocols database file. |

### Errors

If the `setprotoent` routine fails, the error is further specified in the global `errno`.

---

## setservent

Opens and rewinds the network services database file.

### Format

```
#include <netdb.h>

void setservent (int stay_open);
```

### Argument

#### **stay\_open**

Specifies a value used to indicate when to close the network services database file:

- 0 (zero) to close the network services database file after each call to the `getservent` routine.
- Nonzero to keep the network services database file open after each call to `getservent`.

### Description

The `setservent` routine opens the network services database file and resets the file marker to the beginning of the file.

Passing a nonzero value to the *stay\_open* parameter keeps the connection open until you call the `endservent`, or `exit` routine.

See also `endservent`, `exit`, and `getservent` in this appendix.

---

## setsockopt

Sets options on a socket.

### Format

```
#include <socket.h>
```

```
int setsockopt (int s, int level, int optname, char *optval, int optlen); (_DECC_V4_SOURCE)
```

```
int setsockopt (int s, int level, int optname, const void *optval, size_t optlen); (not _DECC_V4_SOURCE)
```

### Arguments

#### **s**

A socket descriptor created by socket.

#### **level**

The protocol level for which the socket options are to be modified. It may have one of the following values:

SOL\_SOCKET

Set the options at the socket level.

p

Any protocol number. Set the options for protocol level p. See the <in.h> header file for the various IPPROTO values.

#### **optname**

Is interpreted by the protocol specified in *level*. Options at each protocol level are documented with the protocol.

The following options are available at the socket level:

SO\_REUSEADDR indicates the rules used in validating addresses supplied in a bind call should allow reuse of local addresses.

SO\_KEEPALIVE keeps connections alive by enabling the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified through an EPIPE error.

SO\_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO\_LINGER delays the internal socket deletion portion of close until either the data has been transmitted, or the device times out (approximately eight minutes).

SO\_BROADCAST is used to enable or disable broadcasting on the socket.

#### **optval**

Points to a buffer containing the parameters of the specified option.

All socket-level options other than SO\_LINGER take an integer parameter that should be nonzero if the option is to be enabled, or zero if it is to be disabled.

SO\_LINGER uses a linger structure parameter defined in the <socket.h> file. This structure specifies the desired state of the option and the linger interval. The option value for the SO\_LINGER command is the address of a linger structure. See the <socket.h> header file for a description of the linger structure.



If the socket promises reliable delivery of data and `l_onoff` is nonzero, the system will block the process on the `close` attempt until it is able to transmit the data or until it decides it is unable to deliver the information. A timeout period, called the linger interval, is specified in `l_linger`.

If `l_onoff` is set to 0 and a `close` is issued, the system will process the `close` in a manner that allows the process to continue as quickly as possible.

#### **optlen**

An integer containing the size of the buffer pointed to by *optval*.

## **Description**

The `setsockopt` routine manipulates options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

When manipulating socket options, the level at which the option resides and the name of the option must be specified. To manipulate options at the socket level, specify *level* as `SOL_SOCKET`. To manipulate options at any other level, the protocol number of the appropriate protocol controlling the option must be supplied. For example, to indicate an option is to be interpreted by the TCP protocol, *level* should be set to the protocol number (`IPPROTO_TCP`) of TCP. See `<in.h>` for the various `IPPROTO` values.

## **Return Values**

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| -1 | Indicates an error; <code>errno</code> is set to one of the following: <ul style="list-style-type: none"> <li>• <code>EBADF</code> – The descriptor is invalid.</li> <li>• <code>ENOTSOCK</code> – The socket descriptor references a file, not a socket.</li> <li>• <code>ENOTCONN</code> – The socket is not connected.</li> <li>• <code>ENOPROTOOPT</code> – The option is unknown.</li> <li>• <code>EFAULT</code> – The <i>optname</i> parameter is not a valid part of the user address space.</li> </ul> |

## shutdown

---

## shutdown

Shuts down all or part of a bidirectional connection on a socket. It can disallow further receives, further sends, or both.

### Format

```
#include <socket.h>

int shutdown (int s, int how);
```

### Arguments

#### **s**

A socket descriptor that is in a connected state as a result of a previous call to either connect or accept.

#### **how**

How the socket is to be shut down. Use one of the following values:

|   |                                                                           |
|---|---------------------------------------------------------------------------|
| 0 | Disallows further calls to <code>recv</code> on the socket.               |
| 1 | Disallows further calls to <code>send</code> on the socket.               |
| 2 | Disallows further calls to both <code>send</code> and <code>recv</code> . |

### Description

The shutdown routine allows communications on a socket to be shut down one piece at a time rather than all at once. It can be used to create unidirectional connections rather than the normal bidirectional (full-duplex) connections.

See also `connect` and `socket` in this appendix.

### Return Values

|    |                                                                                                                                                                                                                                                                                                                                            |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | Indicates success.                                                                                                                                                                                                                                                                                                                         |
| -1 | Indicates an error; <code>errno</code> is set to one of the following: <ul style="list-style-type: none"><li>• <code>EBADF</code> – The socket descriptor is invalid.</li><li>• <code>ENOTSOCK</code> – The descriptor references a file, not a socket.</li><li>• <code>ENOTCONN</code> – The specified socket is not connected.</li></ul> |

---

## socket

Creates an endpoint for communication by returning a special kind of file descriptor called a socket descriptor, which is associated with a *TCP/IP Services for OpenVMS* Socket Device Channel.

### Format

```
#include <socket.h>

int socket (int af, int type, int protocol);
```

### Arguments

#### **af**

The address format to be used in later references to the socket. Addresses specified in subsequent operations using the socket are interpreted according to this format. Currently, only AF\_INET (Internet style) addresses are supported.

#### **type**

The semantics of communication. The type may be SOCK\_STREAM, SOCK\_DGRAM, or SOCK\_RAW.

SOCK\_STREAM type sockets provide sequenced, reliable, two-way connection based byte streams with an available out-of-band data transmission mechanism.

SOCK\_DGRAM sockets support datagrams (connectionless, unreliable data transmission mechanism).

SOCK\_RAW sockets provide access to internal network interfaces, and are available only to users with SYSPRV privilege.

#### **protocol**

The protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type using a given address format. However, it is possible that many protocols may exist, in which case a particular protocol must be specified with this argument. The protocol number to use is particular to the communication domain in which communication is to take place.

### Description

The socket routine provides the primary mechanism for creating sockets. The type and protocol of the socket affect the way the socket behaves and how it can be used.

The operation of sockets is controlled by socket-level options, defined in the file <socket.h>. The setsockopt and getsockopt calls are used to set and get options. Options other than SO\_LINGER take an integer parameter that should be nonzero if the option is to be enabled, or 0 if it is to be disabled. SO\_LINGER uses a linger structure parameter defined in <socket.h>. The members of this structure specify the desired state of the option and the linger interval in the following manner:

SO\_REUSEADDR indicates that the rules used in validating addresses supplied in a bind call should allow reuse of local addresses.

## socket

SO\_KEEPALIVE keeps connections alive by enabling the periodic transmission of messages on a connected socket. Should the connected party fail to respond to these messages, the connection is considered broken and processes using the socket are notified through the error code SS\$\_LINKDISCON.

SO\_DONTROUTE indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.

SO\_LINGER lingers on close if data is present. It controls the actions taken when unsent messages are queued on the socket and a close is performed. When using the setsockopt to set the linger values, the option value for the SO\_LINGER command is the address of a linger structure:

```
struct linger {
 int l_onoff; /* option on/off */
 int l_linger; /* linger time */
};
```

If the socket promises reliable delivery of data and l\_onoff is nonzero, the system will block the process on the attempt until it is able to transmit the data or until it decides it is unable to deliver the information. A timeout period, called the linger interval, is specified in l\_linger. If l\_onoff is set to 0 and a close is issued, the system will process the close in a manner that allows the process to continue as quickly as possible.

SO\_BROADCAST is used to enable or disable broadcasting on the socket.

See also accept, bind, connect, getsockname, getsockopt, listen, read, recv, recvfrom, recvmsg, select, send, sendmsg, sendto, shutdown, and write in this appendix.

## Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | A file descriptor that refers to the socket descriptor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| -1 | Indicates an error; errno is set to one of the following: <ul style="list-style-type: none"><li>• EAFNOSUPPORT – The specified address family is not supported in this version of the system.</li><li>• ESOCKTNOSUPPORT – The specified socket type is not supported in this address family.</li><li>• EPROTONOSUPPORT – The specified protocol is not supported.</li><li>• EPROTOTYPE – Request for a type of socket for which there is no supporting protocol.</li><li>• EMFILE – The per-process descriptor table is full.</li><li>• ENOBUFS – No buffer space is available. The socket cannot be created.</li></ul> |

---

## decc\$socket\_fd

Returns the socket descriptor associated with a Socket Device Channel for direct use with the *Compaq C* RTL.

### Format

```
#include <socket.h>

int decc$socket_fd (int channel);
```

### Argument

**channel**  
A valid Socket Device Channel.

### Description

The `decc$socket_fd` routine associates a valid socket channel with a *Compaq C* RTL file descriptor, and returns the file descriptor. The file descriptor can then be used with one of the *Compaq C* routines that take a file descriptor or socket descriptor as an input parameter.

### Return Values

|    |                                                                |
|----|----------------------------------------------------------------|
| x  | The socket descriptor.                                         |
| −1 | Indicates an error; the socket descriptor cannot be allocated. |

## vaxc\$get\_sdc

---

## vaxc\$get\_sdc

Returns the Socket Device Channel (SDC) associated with a socket descriptor for direct use with the *TCP/IP Services for OpenVMS* product.

### Format

```
#include <socket.h>

short int vaxc$get_sdc (int s);
```

### Argument

**s**  
A socket descriptor.

### Description

The `vaxc$get_sdc` routine returns the SDC associated with a socket. C socket descriptors are normally used either as file descriptors or with one of the routines that takes an explicit socket descriptor as its argument. C sockets are implemented using *TCP/IP Services for OpenVMS* Socket Device Channels. This routine returns the SDC used by a given socket descriptor so that you can use the *TCP/IP Services for OpenVMS* facilities directly by means of various I/O system services (\$QIO).

### Return Values

|   |                                                                 |
|---|-----------------------------------------------------------------|
| 0 | Indicates that <code>s</code> is not an open socket descriptor. |
| x | The SDC number.                                                 |

---

## write

Writes a buffer of data to a socket or file.

### Format

```
#include <unistd.h>

int write (int d, void *buffer, int nbytes);
```

### Arguments

**d**

A descriptor that must refer to a socket or file.

**buffer**

The address of contiguous storage from which the output data is taken.

**nbytes**

The maximum number of bytes involved in the write operation.

### Description

The write routine attempts to write a buffer of data to a socket or file.

See also socket in this appendix.

### Return Values

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x  | The number of bytes written to the socket or file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 0  | Indicates an error.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| -1 | Indicates an error; <code>errno</code> is set to one of the following: <ul style="list-style-type: none"><li>• <code>EBADF</code> – The <i>d</i> argument is not a valid descriptor open for writing.</li><li>• <code>EPIPE</code> – An attempt was made to write to a socket that is not open for reading by any process.</li><li>• <code>EFAULT</code> – Part of the array pointed to by <i>iov</i> or data to be written to the file points outside the process's allocated address space.</li><li>• <code>EWouldBlock</code> – The <code>NBIO</code> (nonblocking) flag is set for the socket descriptor and the process would be delayed in the write operation.</li><li>• <code>EINVAL</code> – The <i>nbytes</i> argument is negative.</li></ul> |

## A.9 Programming Examples

This section provides *Compaq C* socket communications programming examples.

Example A-1 is an example of a TCP/IP Server.

### Example A-1 TCP/IP Server

```
/*=====
*
* Copyright 2002 Compaq Information Technologies Group, L.P.
*
* Compaq and the Compaq logo are trademarks of Compaq Information Technologies
* Group, L.P. in the U.S. and/or other countries.
*
* Confidential computer software. Valid license from Compaq required for
* possession, use or copying. Consistent with FAR 12.211 and 12.212,
* Commercial Computer Software, Computer Software Documentation, and
* Technical Data for Commercial Items are licensed to the U.S. Government
* under vendor's standard commercial license.
*
*
*
* FACILITY:
* INSTALL
*
*
* ABSTRACT:
* This is an example of a TCP/IP server using the IPC
* socket interface.
*
*
* ENVIRONMENT:
* UCX V1.2 or higher, VMS V5.2 or higher
*
* This example is portable to UNIX. The include
* files are conditionally defined for both systems, and
* "perror" is used for error reporting.
* BUILD INSTRUCTIONS:
*
* To link in VAXC/VMS you must have the following
* entries in your .opt file:
* sys$library:ucx$ipc.olb/lib
* sys$share:vaxcrtl.exe/share
*
* For Compaq C or Compaq C++, compile /PREFIX=ALL and link via
* $ link UCX$TCP_SERVER_IPC
*
* To build this example program, use commands of the following form:
*
* using the Compaq C compiler:
*
* $ cc/prefix=all UCX$TCP_SERVER_IPC.C
* $ link UCX$TCP_SERVER_IPC
*
* using the Compaq C++ compiler:
*
* $ cxx/prefix=all/define=VMS UCX$TCP_SERVER_IPC.C
* $ link UCX$TCP_SERVER_IPC
```

(continued on next page)



### Example A-1 (Cont.) TCP/IP Server

```
*
* using the VAX C compiler:
*
* $ cc /vaxc UCX$TCP_SERVER_IPC.C
* $ link UCX$TCP_SERVER_IPC, -
* SYS$LIBRARY:UCX$IPC/LIB, -
* SYS$INPUT/OPTIONS
* SYS$SHARE:UCX$IPC_SHR/SHARE
* SYS$SHARE:VAXCRTL.EXE/SHARE
*
* AUTHORS:
* UCX Developer
*
* CREATION DATE: May 23, 1989
*
* MODIFICATION HISTORY:
*
* 16 May 1996 Joseph J. Vlcek
* Make compatible with the Compaq C and Compaq C++ compilers.
* Add directions on how to build this example modules.
*
*/

/*
*
* INCLUDE FILES
*
*/

#ifdef VMS
#include <descrip.h> /* VMS descriptor stuff */
#include <in.h> /* internet system Constants and structures. */
#include <inet.h> /* Network address info. */
#include <iodef.h> /* I/O FUNCTION CODE DEFS */
#include <lib$routines.h> /* LIB$ RTL-routine signatures. */
#include <netdb.h> /* Network database library info. */
#include <signal.h> /* UNIX style Signal Value Definitions */
#include <socket.h> /* TCP/IP socket definitions. */
#include <ssdef.h> /* SS$_xyz sys ser return statistics */
#include <starlet.h> /* Sys ser calls */
#include <stdio.h> /* UNIX 'Standard I/O' Definitions */
#include <stdlib.h> /* General Utilities */
#include <string.h> /* String handling function definitions */
#include <ucx$inetdef.h> /* UCX network definitions */
#include <unixio.h> /* Prototypes for UNIX emulation functions */
#else
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#endif
#endif
```

(continued on next page)

### Example A-1 (Cont.) TCP/IP Server

```
/*
 * Functional Description
 *
 * This example creates a socket of type SOCK_STREAM (TCP),
 * binds and listens on the socket, receives a message,
 * and closes the connection.
 * Error messages are printed to the screen.
 *
 * IPC calls used:
 * accept
 * bind
 * close
 * gethostbyname
 * listen
 * recv
 * shutdown
 * socket
 *
 * Formal Parameters
 * The server program expects one parameter:
 * portnumber ... port number where it will listen
 *
 * Routine Value
 *
 * Status
 */

/*-----*/
cleanup(int how_many, int sock1, int sock2)
{
 int retval;

 /*
 * Shutdown and close sock1 completely.
 */
 retval = shutdown(sock1,2);
 if (retval == -1)
 perror ("shutdown");
 retval = close (sock1);
 if (retval)
 perror ("close");

 /*
 * If given, shutdown and close sock2.
 */
 if (how_many == 2)
 {
 retval = shutdown(sock2,2);
 if (retval == -1)
 perror ("shutdown");
 retval = close (sock2);
 if (retval)
 perror ("close");
 }

 exit(1);
} /* end cleanup*/
```

(continued on next page)

### Example A-1 (Cont.) TCP/IP Server

```
/*-----*/
main(int argc, char **argv)
{
 int sock_2, sock_3; /* sockets */
 static char message[BUFSIZ];
 static struct sockaddr_in sock2_name; /* Address struct for socket2.*/
 static struct sockaddr_in retsock2_name; /* Address struct for socket2.*/
 struct hostent hostentstruct; /* Storage for hostent data. */
 struct hostent *hostentptr; /* Pointer to hostent data. */
 static char hostname[256]; /* Name of local host. */
 int flag;
 int retval; /* helpful for debugging */
 int namelength;

 /*
 * Check input parameters.
 */
 if (argc != 2)
 {
 printf("Usage: server portnumber.\n");
 exit(1);
 }

 /*
 * Open socket 2: AF_INET, SOCK_STREAM.
 */
 if ((sock_2 = socket (AF_INET, SOCK_STREAM, 0)) == -1)
 {
 perror("socket");
 exit(1);
 }

 /*
 * Get the host local name.
 */
 retval = gethostname(hostname,sizeof hostname);
 if (retval)
 {
 perror ("gethostname");
 cleanup (1, sock_2, 0);
 }

 /*
 * Get pointer to network data structure for socket 2.
 */
 if ((hostentptr = gethostbyname (hostname)) == NULL)
 {
 perror("gethostbyname");
 cleanup(1, sock_2, 0);
 }

 /*
 * Copy hostent data to safe storage.
 */
 hostentstruct = *hostentptr;

 /*
 * Fill in the name & address structure for socket 2.
 */
 sock2_name.sin_family = hostentstruct.h_addrtype;
 sock2_name.sin_port = htons(atoi(argv[1]));
 sock2_name.sin_addr = * ((struct in_addr *) hostentstruct.h_addr);
}
```

(continued on next page)

### Example A-1 (Cont.) TCP/IP Server

```
/*
 * Bind name to socket 2.
 */
retval = bind (sock_2,
 (struct sockaddr*)&sock2_name,
 sizeof sock2_name);
if (retval)
{
 perror("bind");
 cleanup(1, sock_2, 0);
}

/*
 * Listen on socket 2 for connections.
 */
retval = listen (sock_2, 5);
if (retval)
{
 perror("listen");
 cleanup(1, sock_2, 0);
}

/*
 * Accept connection from socket 2:
 * accepted connection will be on socket 3.
 */
namelength = sizeof (sock2_name);
sock_3 = accept (sock_2, (struct sockaddr*)&sock2_name, &namelength);
if (sock_3 == -1)
{
 perror ("accept");
 cleanup(2, sock_2, sock_3);
}

/*
 * Receive message from socket 1.
 */
flag = 0; /* maybe 0 or MSG_OOB or MSG_PEEK */
retval = recv(sock_3, message, sizeof (message), flag);
if (retval == -1)
{
 perror ("receive");
 cleanup(2, sock_2, sock_3);
}
else
 printf (" %s\n", message);

/*
 * Call cleanup to shutdown and close sockets.
 */
cleanup(2, sock_2, sock_3);
} /* end main */
```

Example A-2 is an example of a TCP/IP Client.

### Example A-2 TCP/IP Client

```
/*=====
*
* Copyright 2002 Compaq Information Technologies Group, L.P.
*
* Compaq and the Compaq logo are trademarks of Compaq Information Technologies
* Group, L.P. in the U.S. and/or other countries.
*
* Confidential computer software. Valid license from Compaq required for
* possession, use or copying. Consistent with FAR 12.211 and 12.212,
* Commercial Computer Software, Computer Software Documentation, and
* Technical Data for Commercial Items are licensed to the U.S. Government
* under vendor's standard commercial license.
*
*
*
* FACILITY:
* INSTALL
*
*
* ABSTRACT:
* This is an example of a TCP/IP client using the IPC
* socket interface.
*
*
* ENVIRONMENT:
* UCX V1.2 or higher, VMS V5.2 or higher
*
* This example is portable to UNIX. The include
* files are conditionally defined for both systems, and
* "perror" is used for error reporting.
* BUILD INSTRUCTIONS:
*
* To link in VAXC/VMS you must have the following
* entries in your .opt file:
* sys$library:ucx$ipc.olb/lib
* sys$share:vaxcrtl.exe/share
*
* For Compaq C or Compaq C++, compile /PREFIX=ALL and link via
* $ link UCX$TCP_CLIENT_IPC
*
* To build this example program, use commands of the following form:
*
* using the Compaq C compiler:
*
* $ cc/prefix=all UCX$TCP_CLIENT_IPC.C
* $ link UCX$TCP_CLIENT_IPC
*
* using the Compaq C++ compiler:
*
* $ cxx/prefix=all/define=VMS UCX$TCP_CLIENT_IPC.C
* $ link UCX$TCP_CLIENT_IPC
*
```

(continued on next page)

## Example A-2 (Cont.) TCP/IP Client

```
* using the VAX C compiler:
*
* $ cc /vaxc UCX$TCP_CLIENT_IPC.C
* $ link UCX$TCP_CLIENT_IPC, -
* SYS$LIBRARY:UCX$IPC/LIB, -
* SYS$INPUT/OPTIONS
* SYS$SHARE:UCX$IPC_SHR/SHARE
*
* AUTHORS:
* UCX Developer
*
* CREATION DATE: May 23, 1989
*
* MODIFICATION HISTORY:
*
* 16 May 1996 Joseph J. Vlcek
* Make compatible with the Compaq C and Compaq C++ compilers.
* Add directions on how to build this example modules.
*
*/

/*
* INCLUDE FILES
*
*/

#if defined(VMS) || defined(__VMS)
#include <stdlib.h>
#include <unixio.h>
#include <errno.h>
#include <types.h>
#include <stdio.h>
#include <socket.h>
#include <in.h>
#include <netdb.h> /* change hostent to comply with BSD 4.3*/
#include <inet.h>
#include <ucx$inetdef.h> /* INET symbol definitions */
#else
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#endif

/*
*
* MACRO DEFINITIONS
*
*/

#ifndef vms
#define TRUE 1
#define FALSE 0
#endif

void cleanup(int shut, int socket);
```

(continued on next page)

## Example A-2 (Cont.) TCP/IP Client

```
/*
 * Functional Description
 *
 * This example creates a socket of type SOCK_STREAM (TCP),
 * initiates a connection to the remote host, sends
 * a message to the remote host, and closes the connection.
 * Error messages are printed to the screen.
 *
 * IPC calls used:
 * close
 * connect
 * gethostbyname
 * send
 * shutdown
 * socket
 *
 * Formal Parameters
 * The client program expects two parameters:
 * hostname ... name of remote host
 * portnumber ... port where remote host(server) is listening
 *
 * Routine Value
 *
 * Status
 */
/*-----*/
main(int argc, char **argv)
{
 int sock_1; /* socket */
 static char message[] = "Hi there.";
 static struct sockaddr_in sock2_name; /* Address struct for socket2.*/
 struct hostent hostentstruct; /* Storage for hostent data. */
 struct hostent *hostentptr; /* Pointer to hostent data. */
 static char hostname[256]; /* Name of local host. */
 int flag;
 int retval; /* Helpful for debugging. */
 int shut = FALSE; /* Flag to cleanup. */

 /*
 * Check input parameters.
 */
 if (argc != 3)
 {
 printf("Usage: client hostname portnumber.\n");
 exit(EXIT_FAILURE);
 }

 /*
 * Open socket 1: AF_INET, SOCK_STREAM.
 */
 if ((sock_1 = socket (AF_INET, SOCK_STREAM, 0)) == -1)
 {
 perror("socket");
 exit(EXIT_FAILURE);
 }
}
```

(continued on next page)

### Example A-2 (Cont.) TCP/IP Client

```
/*
 *Get pointer to network data structure for socket 2 (remote host).
 */
if ((hostentptr = gethostbyname (argv[1])) == NULL)
{
 perror("gethostbyname");
 cleanup(shut, sock_1);
}

/*
 * Copy hostent data to safe storage.
 */
hostentstruct = *hostentptr;

/*
 * Fill in the name & address structure for socket 2.
 */
sock2_name.sin_family = hostentstruct.h_addrtype;
sock2_name.sin_port = htons(atoi(argv[2]));
sock2_name.sin_addr = * ((struct in_addr *) hostentstruct.h_addr);

/*
 * Connect socket 1 to sock2_name.
 */
retval = connect(sock_1, (struct sockaddr *)&sock2_name,
 sizeof (sock2_name));
if (retval)
{
 perror("connect");
 cleanup(shut, sock_1);
}

/*
 * Send message to socket 2.
 */
flag = 0; /* maybe 0 or MSG_OOB */
retval = send(sock_1, message ,sizeof (message), flag);
if (retval < 0)
{
 perror ("send");
 shut = TRUE;
}

/*
 * Call cleanup to shutdown and close socket.
 */
cleanup(shut, sock_1);
} /* end main */
```

(continued on next page)



### Example A-2 (Cont.) TCP/IP Client

```
/*-----*/
void cleanup(int shut, int socket)
{
 int retval;

 /*
 * Shutdown socket completely -- only if it was connected.
 */
 if (shut) {
 retval = shutdown(socket,2);
 if (retval == -1)
 perror ("shutdown");
 }

 /*
 * Close socket.
 */
 retval = close (socket);
 if (retval)
 perror ("close");
 exit(EXIT_SUCCESS);
} /* end main */
```

Example A-3 is an example of a UDP/IP Server.

### Example A-3 UDP/IP Server

```
/*=====
*
* Copyright 2002 Compaq Information Technologies Group, L.P.
*
* Compaq and the Compaq logo are trademarks of Compaq Information Technologies
* Group, L.P. in the U.S. and/or other countries.
*
* Confidential computer software. Valid license from Compaq required for
* possession, use or copying. Consistent with FAR 12.211 and 12.212,
* Commercial Computer Software, Computer Software Documentation, and
* Technical Data for Commercial Items are licensed to the U.S. Government
* under vendor's standard commercial license.
*
*
*
* FACILITY:
* INSTALL
*
*
* ABSTRACT:
* This is an example of a UDP/IP server using the IPC
* socket interface.
*
*
* ENVIRONMENT:
* UCX V1.2 or higher, VMS V5.2 or higher
*
* This example is portable to UNIX. The include
* files are conditionally defined for both systems, and
* "perror" is used for error reporting.
* BUILD INSTRUCTIONS:
*
* To link in VAXC/VMS you must have the following
* entries in your .opt file:
* sys$library:ucx$ipc.olb/lib
* sys$share:vaxcrtl.exe/share
*
* For Compaq C or Compaq C++, compile /PREFIX=ALL and link via
* $ link UCX$UDP_SERVER_IPC
*
* To build this example program, use commands of the following form:
*
* using the Compaq C compiler:
*
* $ cc/prefix=all UCX$UDP_SERVER_IPC.C
* $ link UCX$UDP_SERVER_IPC
*
* using the Compaq C++ compiler:
*
* $ cxx/prefix=all/define=VMS UCX$UDP_SERVER_IPC.C
* $ link UCX$UDP_SERVER_IPC
```

(continued on next page)

### Example A-3 (Cont.) UDP/IP Server

```
* using the VAX C compiler:
*
* $ cc /vaxc UCX$UDP_SERVER_IPC.C
* $ link UCX$UDP_SERVER_IPC, -
* SYS$LIBRARY:UCX$IPC/LIB, -
* SYS$INPUT/OPTIONS
* SYS$SHARE:UCX$IPC_SHR/SHARE
* SYS$SHARE:VAXCTRL.EXE/SHARE
*
*
* AUTHORS:
* UCX Developer
*
* CREATION DATE: May 23, 1989
*
* MODIFICATION HISTORY:
*
* 16 May 1996 Joseph J. Vlcek
* Make compatible with the Compaq C and Compaq C++ compilers.
* Add directions on how to build this example modules.
*
*/

/*
*
* INCLUDE FILES
*
*/

#ifdef VMS
#include <descrip.h> /* VMS descriptor stuff */
#include <errno.h> /* Unix style error codes for IO routines. */
#include <in.h> /* internet system Constants and structures. */
#include <inet.h> /* Network address info. */
#include <iodef.h> /* I/O FUNCTION CODE DEFS */
#include <lib$routines.h> /* LIB$ RTL-routine signatures. */
#include <netdb.h> /* Network database library info. */
#include <signal.h> /* UNIX style Signal Value Definitions */
#include <socket.h> /* TCP/IP socket definitions. */
#include <ssdef.h> /* SS$_xyz sys ser return statistics */
#include <starlet.h> /* Sys_ser calls */
#include <stdio.h> /* UNIX 'Standard I/O' Definitions */
#include <stdlib.h> /* General Utilities */
#include <string.h> /* String handling function definitions */
#include <ucx$inetdef.h> /* UCX network definitions */
#include <unixio.h> /* Prototypes for UNIX emulation functions */
#else
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#include <time.h> /* timeval declared here */
#endif

cleanup(int socket)
{
 int retval;
```

(continued on next page)

### Example A-3 (Cont.) UDP/IP Server

```
/*
 * Shutdown socket completely.
 */
retval = shutdown(socket,2);
if (retval == -1)
 perror ("shutdown");

/*
 * Close socket.
 */
retval = close (socket);
if (retval)
 perror ("close");

exit(1);
} /* end cleanup */

/*
 * Functional Description
 *
 * This example creates a socket of type SOCK_DGRAM (UDP), binds
 * it, and selects to receive a message on the socket.
 * Error messages are printed to the screen.
 *
 * IPC calls used:
 * bind
 * close
 * gethostbyname
 * recvfrom
 * select
 * shutdown
 * socket
 *
 * Formal Parameters
 * The server program expects one parameter:
 * portnumber ... port where it is listening
 *
 * Routine Value
 *
 * Status
 */

/*-----*/
main(int argc, char **argv)
{
 int rmask, wmask, emask;
 int sock_2; /* Socket2 descriptor. */
 int buflen, fromlen;
 char recvbuf[BUFSIZ];
 static struct sockaddr_in sock1_name; /* Address struct for socket1.*/
 static struct sockaddr_in sock2_name; /* Address struct for socket2.*/
 int namelength;
 struct hostent hostentstruct; /* Storage for hostent data. */
 struct hostent *hostentptr; /* Pointer to hostent data. */
 static char hostname[256]; /* Name of local host. */
 int retval;
 int flag;
 struct timeval timeout;
```

(continued on next page)

### Example A-3 (Cont.) UDP/IP Server

```
/*
 * Check input parameters.
 */
if (argc != 2)
{
 printf("Usage: server portnumber.\n");
 exit(1);
}

/*
 * Open socket 2: AF_INET, SOCK_DGRAM.
 */
if ((sock_2 = socket (AF_INET, SOCK_DGRAM, 0)) == -1)
{
 perror("socket");
 exit(1);
}

/*
 * Get the local host name.
 */
retval = gethostname(hostname, sizeof hostname);
if (retval)
{
 perror ("gethostname");
 cleanup(sock_2);
}

/*
 * Get pointer to network data structure for local host.
 */
if ((hostentptr = gethostbyname (hostname)) == NULL)
{
 perror("gethostbyname");
 cleanup(sock_2);
}

/*
 * Copy hostent data to safe storage.
 */
hostentstruct = *hostentptr;

/*
 * Fill in the address structure for socket 2.
 */
sock2_name.sin_family = hostentstruct.h_addrtype;
sock2_name.sin_port = htons(atoi(argv[1]));
sock2_name.sin_addr = * ((struct in_addr *) hostentstruct.h_addr);
```

(continued on next page)

### Example A-3 (Cont.) UDP/IP Server

```
/*
 * Bind name to socket 2.
 */
retval = bind (sock_2,
 (struct sockaddr*)&sock2_name,
 sizeof sock2_name);

if (retval)
{
 perror("bind");
 cleanup(sock_2);
}

/*
 * Select socket to receive message.
 */
emask = wmask = 0;
rmask = (1<<sock_2); /* set read mask */
timeout.tv_sec = 30;
timeout.tv_usec = 0;

retval = select(32, &rmask, &wmask, &emask, &timeout);
switch (retval)
{
 case -1:
 {
 perror("select");
 cleanup(sock_2);
 }
 case 0:
 {
 printf("Select timed out with status 0.\n");
 cleanup(sock_2);
 }
 default:
 if ((rmask & (1<<sock_2)) == 0)
 {
 printf("Select not reading on sock_2.\n");
 cleanup(sock_2);
 }
} /*switch*/

/*
 * Recvfrom buffer - from sock1 on sock2.
 */
buflen = sizeof(recvbuf);
fromlen = sizeof(sock1_name);
flag = 0; /* flag may be MSG_OOB and/or MSG_PEEK */
retval = recvfrom(sock_2,
 recvbuf,
 buflen,
 flag,
 (struct sockaddr*)&sock1_name,
 &fromlen);

if (retval == -1)
 perror("recvfrom");
else
 printf (" %s\n", recvbuf);
```

(continued on next page)

### **Example A-3 (Cont.) UDP/IP Server**

```
 /*
 * Call cleanup to shut down and close socket.
 */
 cleanup(sock_2);
} /* end main */
```

Example A-4 is an example of a UDP/IP client.

#### Example A-4 UDP/IP Client

```
/*=====
*
* Copyright 2002 Compaq Information Technologies Group, L.P.
*
* Compaq and the Compaq logo are trademarks of Compaq Information Technologies
* Group, L.P. in the U.S. and/or other countries.
*
* Confidential computer software. Valid license from Compaq required for
* possession, use or copying. Consistent with FAR 12.211 and 12.212,
* Commercial Computer Software, Computer Software Documentation, and
* Technical Data for Commercial Items are licensed to the U.S. Government
* under vendor's standard commercial license.
*
*
*
* FACILITY:
* INSTALL
*
*
* ABSTRACT:
* This is an example of a UDP/IP client using the IPC
* socket interface.
*
*
* ENVIRONMENT:
* UCX V1.2 or higher, VMS V5.2 or higher
*
* This example is portable to UNIX. The include
* files are conditionally defined for both systems, and
* "perror" is used for error reporting.
*
* BUILD INSTRUCTIONS:
*
* To link in VAXC/VMS you must have the following
* entries in your .opt file:
* sys$library:ucx$ipc.olb/lib
* sys$share:vaxcrtl.exe/share
*
* For Compaq C or Compaq C++, compile /PREFIX=ALL and link via
* $ link UCX$UDP_CLIENT_IPC
*
* To build this example program, use commands of the following form:
*
* using the Compaq C compiler:
*
* $ cc/prefix=all UCX$UDP_CLIENT_IPC.C
* $ link UCX$UDP_CLIENT_IPC
*
* using the Compaq C++ compiler:
*
* $ cxx/prefix=all/define=VMS UCX$UDP_CLIENT_IPC.C
* $ link UCX$UDP_CLIENT_IPC
```

(continued on next page)



#### Example A-4 (Cont.) UDP/IP Client

```
* using the VAX C compiler:
*
* $ cc /vaxc UCX$UDP_CLIENT_IPC.C
* $ link UCX$UDP_CLIENT_IPC, -
* SYS$LIBRARY:UCX$IPC/LIB, -
* SYS$INPUT/OPTIONS
* SYS$SHARE:UCX$IPC_SHR/SHARE
* SYS$SHARE:VAXCTRL.EXE/SHARE
*
* AUTHORS:
* UCX Developer
*
* CREATION DATE: May 23, 1989
*
* MODIFICATION HISTORY:
*
* 16 May 1996 Joseph J. Vlcek
* Make compatible with the Compaq C and Compaq C++ compilers.
* Add directions on how to build this example modules.
*/

/*
*
* INCLUDE FILES
*
*/

#ifdef VMS
#include <descrip.h> /* VMS descriptor stuff */
#include <errno.h> /* Unix style error codes for IO routines. */
#include <in.h> /* internet system Constants and structures. */
#include <inet.h> /* Network address info. */
#include <iodef.h> /* I/O FUNCTION CODE DEFS */
#include <lib$routines.h> /* LIB$ RTL-routine signatures. */
#include <netdb.h> /* Network database library info. */
#include <signal.h> /* UNIX style Signal Value Definitions */
#include <socket.h> /* TCP/IP socket definitions. */
#include <ssdef.h> /* SS$ <xyz> sys ser return statistics */
#include <starlet.h> /* Sys_ser calls */
#include <stdio.h> /* UNIX 'Standard I/O' Definitions */
#include <stdlib.h> /* General Utilities */
#include <string.h> /* String handling function definitions */
#include <ucx$inetdef.h> /* UCX network definitions */
#include <unixio.h> /* Prototypes for UNIX emulation functions */
#else
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/uio.h>
#endif
```

(continued on next page)

#### Example A-4 (Cont.) UDP/IP Client

```
/*-----*/
cleanup(int socket)
{
 int retval;

 /*
 * Shutdown socket completely.
 */
 retval = shutdown(socket,2);
 if (retval == -1)
 perror ("shutdown");

 /*
 * Close socket.
 */
 retval = close (socket);
 if (retval)
 perror ("close");

 exit(1);
} /* end cleanup */

/*
 * Functional Description
 *
 * This example creates a socket of type SOCK_DGRAM (UDP),
 * binds it, and sends a message to the given host and port number.
 * Error messages are printed to the screen.
 *
 * IPC calls used:
 * bind
 * close
 * gethostbyname
 * sendto
 * shutdown
 * socket
 *
 * Formal Parameters
 * The client program expects two parameters:
 * hostname ... name of remote host
 * portnumber ... port where remote host(server) is listening
 *
 * Routine Value
 *
 * Status
 */
```

(continued on next page)

#### Example A-4 (Cont.) UDP/IP Client

```
/*-----*/
main(int argc, char **argv)
{
 int sock_1; /* Socket 1 descriptor. */
 int sendlen, tolen;
 static char sendbuf[] = "Hi there.";
 static struct sockaddr_in sock2_name; /* Address struct for socket2.*/
 int namelength;
 struct hostent hostentstruct; /* Storage for hostent data. */
 struct hostent *hostentptr; /* Pointer to hostent data. */
 static char hostname[256]; /* Name of local host. */
 int flag;
 int retval;

 /*
 * Check input parameters.
 */
 if (argc != 3)
 {
 printf("Usage: client hostname portnumber.\n");
 exit(1);
 }

 /*
 * Open socket 1: AF_INET, SOCK_DGRAM.
 */
 if ((sock_1 = socket (AF_INET, SOCK_DGRAM, 0)) == -1)
 {
 perror("socket");
 exit(1);
 }

 /*
 * Get pointer to network data structure for given host.
 */
 if ((hostentptr = gethostbyname (argv[1])) == NULL)
 {
 perror("gethostbyname");
 cleanup(sock_1);
 }

 /*
 * Copy hostent data to safe storage.
 */
 hostentstruct = *hostentptr;

 /*
 * Fill in the address structure for socket 2 (to receive message).
 */
 sock2_name.sin_family = hostentstruct.h_addrtype;
 sock2_name.sin_port = htons(atoi(argv[2]));
 sock2_name.sin_addr = * ((struct in_addr *) hostentstruct.h_addr);
}
```

(continued on next page)

#### Example A-4 (Cont.) UDP/IP Client

```
/*
 * Initialize send block.
 */
sendlen = sizeof sendbuf;
tolen = sizeof sock2_name;
flag = 0; /* flag may be MSG_OOB */

/*
 * Send message from socket 1 to socket 2.
 */
retval = sendto(sock_1, sendbuf, sendlen, flag,
 (struct sockaddr*)&sock2_name, tolen);
if (retval == -1)
{
 perror ("sendto");
 cleanup(sock_1);
}

/*
 * Call cleanup to shut down and close socket.
 */
cleanup(sock_1);
} /* end main */
```

## Version-Dependency Tables

New functions are added to the *Compaq C* Run-Time Library with each version of *Compaq C*. These functions are implemented and shipped with the *OpenVMS* operating system, while the documentation and header files containing their prototypes are shipped with versions of the *Compaq C* compiler.

You might have a newer version of *Compaq C* that has header files and documentation for functions that are not supported on your older *OpenVMS* system. For example, if your target operating system platform is *OpenVMS* Version 7.2, you cannot use *Compaq C* RTL functions introduced on *OpenVMS* Version 7.3, even though they are documented in this manual.

This appendix contains several tables that list what *Compaq C* RTL functions are supported on recent *OpenVMS* versions. This is helpful for determining the functions to avoid using on your target *OpenVMS* platforms.

Also, for *Compaq C* and C++ Version 5.6 and higher, a C RTL backport object library is included with the compiler distribution kit. The backport object library allows developers on older versions of *OpenVMS* to use the latest C run-time library functions. For more information, see the file `SYS$LIBRARY:DECC$CTRL.README` on your system.

### B.1 Functions Available on all *OpenVMS VAX* and *OpenVMS Alpha* Versions

Table B–1 lists functions available on all *OpenVMS VAX* and *OpenVMS Alpha* versions.

**Table B–1 Functions Available on All *OpenVMS* Systems**

|                    |                        |                  |                       |
|--------------------|------------------------|------------------|-----------------------|
| abort              | abs                    | access           | acos                  |
| alarm              | asctime                | asin             | assert                |
| atan2              | atan                   | atexit           | atof                  |
| atoi               | atoll ( <i>Alpha</i> ) | atol             | atoq ( <i>Alpha</i> ) |
| box                | brk                    | bsearch          | cabs                  |
| calloc             | ceil                   | cfree            | chdir                 |
| chmod              | chown                  | clearerr         | clock                 |
| close              | cosh                   | cos              | creat                 |
| ctermid            | ctime                  | cuserid          | decc\$ctrl_init       |
| decc\$fix_time     | decc\$from_vms         | decc\$match_wild | decc\$record_read     |
| decc\$record_write | decc\$set_reentrancy   | decc\$to_vms     | decc\$translate_vms   |

(continued on next page)

**Table B–1 (Cont.) Functions Available on All OpenVMS Systems**

|                        |                         |            |                       |
|------------------------|-------------------------|------------|-----------------------|
| delete                 | delwin                  | difftime   | div                   |
| dup2                   | dup                     | ecvt       | endwin                |
| execle                 | execlp                  | execl      | execve                |
| execvp                 | execv                   | exit       | _exit                 |
| exp                    | fabs                    | fclose     | fcvt                  |
| fdopen                 | feof                    | ferror     | fflush                |
| fgetc                  | fgetname                | fgetpos    | fgets                 |
| fileno                 | floor                   | fmod       | fopen                 |
| fprintf                | fputc                   | fputs      | fread                 |
| free                   | freopen                 | frexp      | fscanf                |
| fseek                  | fsetpos                 | fstat      | fsync                 |
| ftell                  | ftime                   | fwait      | fwrite                |
| gcvt                   | getchar                 | getcwd     | getc                  |
| getegid                | getenv                  | geteuid    | getgid                |
| getname                | getpid                  | getppid    | gets                  |
| getuid                 | getw                    | gmtime     | gsignal               |
| hypot                  | initscr                 | isalnum    | isalpha               |
| isapipe                | isascii                 | isatty     | iscntrl               |
| isdigit                | isgraph                 | islower    | isprint               |
| ispunct                | isspace                 | isupper    | isxdigit              |
| kill                   | labs                    | ldexp      | ldiv                  |
| llabs ( <i>Alpha</i> ) | lldiv ( <i>Alpha</i> )  | localeconv | localtime             |
| log10                  | log                     | longjmp    | longname              |
| lseek                  | lwait                   | malloc     | mblen                 |
| mbstowcs               | mbtowc                  | memchr     | memcmp                |
| memcpy                 | memmove                 | memset     | mkdir                 |
| mktemp                 | mktime                  | modf       | mvwin                 |
| mv[w]addstr            | newwin                  | nice       | open                  |
| overlay                | overwrite               | pause      | perror                |
| pipe                   | pow                     | printf     | putchar               |
| putc                   | puts                    | putw       | qabs ( <i>Alpha</i> ) |
| qdiv ( <i>Alpha</i> )  | qsort                   | raise      | rand                  |
| read                   | realloc                 | remove     | rename                |
| rewind                 | sbrk                    | scanf      | scroll                |
| setbuf                 | setgid                  | setjmp     | setlocale             |
| setuid                 | setvbuf                 | sigblock   | signal                |
| sigpause               | sigstack ( <i>VAX</i> ) | sigvec     | sinh                  |
| sin                    | sleep                   | sprintf    | sqrt                  |
| srand                  | sscanf                  | ssignal    | stat                  |

(continued on next page)

**Table B–1 (Cont.) Functions Available on All *OpenVMS* Systems**

|                         |                           |                          |                          |
|-------------------------|---------------------------|--------------------------|--------------------------|
| strcat                  | strchr                    | strcmp                   | strcoll                  |
| strcpy                  | strcspn                   | strerror                 | strftime                 |
| strlen                  | strncat                   | strncmp                  | strncpy                  |
| strpbrk                 | strrchr                   | strspn                   | strstr                   |
| strtod                  | strtok                    | strtoll ( <i>Alpha</i> ) | strtol                   |
| strtoq ( <i>Alpha</i> ) | strtoull ( <i>Alpha</i> ) | strtoul                  | strtouq ( <i>Alpha</i> ) |
| strxfrm                 | subwin                    | system                   | tanh                     |
| tan                     | times                     | time                     | tmpfile                  |
| tmpnam                  | toascii                   | tolower                  | _tolower                 |
| touchwin                | toupper                   | _toupper                 | ttyname                  |
| umask                   | ungetc                    | vaxc\$calloc_opt         | vaxc\$cfree_opt          |
| vaxc\$ctrl_init         | vaxc\$establish           | vaxc\$free_opt           | vaxc\$malloc_opt         |
| vaxc\$realloc_opt       | va_arg                    | va_count                 | va_end                   |
| va_start                | va_start_1                | vfork                    | vfprintf                 |
| vprintf                 | vsprintf                  | wait                     | wcstombs                 |
| wctomb                  | write                     | [w]addch                 | [w]addstr                |
| [w]clear                | [w]clrattr                | [w]clrtoebot             | [w]clrtoeol              |
| [w]delch                | [w]deleteln               | [w]erase                 | [w]getch                 |
| [w]getstr               | [w]inch                   | [w]insch                 | [w]insertln              |
| [w]insstr               | [w]move                   | [w]printw                | [w]refresh               |
| [w]scanw                | [w]setattr                | [w]standend              | [w]standout              |

## B.2 Functions Available on *OpenVMS* Version 6.2 and Higher

Table B–2 lists functions available on *OpenVMS* VAX and *OpenVMS* Alpha Version 6.2 and higher.

**Table B–2 Functions Added in *OpenVMS* Version 6.2**

|            |             |          |             |
|------------|-------------|----------|-------------|
| catclose   | catgets     | catopen  | fgetwc      |
| fgetws     | fputwc      | fputws   | getopt      |
| getwc      | getwchar    | iconv    | iconv_close |
| iconv_open | iswalnum    | iswalpha | iswcntrl    |
| iswctype   | iswdigit    | iswgraph | iswlower    |
| iswprint   | iswpunct    | iswspace | iswupper    |
| iswxdigit  | nl_langinfo | putwc    | putwchar    |
| strnlen    | strptime    | tolower  | towupper    |
| ungetwc    | wscat       | wcschr   | wcscmp      |
| wcscoll    | wscpy       | wcscspn  | wcsftime    |
| wcslen     | wcsncat     | wcsncmp  | wcsncpy     |

(continued on next page)

**Table B–2 (Cont.) Functions Added in *OpenVMS* Version 6.2**

|         |         |          |         |
|---------|---------|----------|---------|
| wcspbrk | wcsrchr | wcsspn   | wcstol  |
| wcstoul | wcswcs  | wcswidth | wcsxfrm |
| wcstod  | wctype  | wcwidth  | wcstok  |

### B.3 Functions Available on *OpenVMS* Version 7.0 and Higher

Table B–3 lists functions available on *OpenVMS VAX* and *OpenVMS Alpha* Version 7.0 and higher.

**Table B–3 Functions Added in *OpenVMS* Version 7.0**

|              |             |               |            |
|--------------|-------------|---------------|------------|
| basename     | bcmp        | bcopy         | btowc      |
| bzero        | closedir    | confstr       | dirname    |
| drand48      | erand48     | ffs           | fpathconf  |
| ftruncate    | ftw         | fwide         | fwprintf   |
| fwscanf      | getclock    | getdtablesize | getitimer  |
| getlogin     | getpagesize | getpwnam      | getpwuid   |
| gettimeofday | index       | initstate     | jrand48    |
| lcong48      | lrand48     | mbrlen        | mbrtowc    |
| mbsinit      | mbsrtowcs   | memccpy       | mkstemp    |
| mmap         | mprotect    | mrnd48        | msync      |
| munmap       | nrnd48      | opendir       | pathconf   |
| pclose       | popen       | putenv        | random     |
| readdir      | rewinddir   | rindex        | rmdir      |
| seed48       | seekdir     | setenv        | setitimer  |
| setstate     | sigaction   | sigaddset     | sigdelset  |
| sigemptyset  | sigfillset  | sigismember   | siglongjmp |
| sigpending   | sigprocmask | sigsetjmp     | sigsuspend |
| srand48      | srandom     | strcasecmp    | strdup     |
| strfmon      | strncasecmp | strsep        | swab       |
| swprintf     | swscanf     | sysconf       | telldir    |
| tempnam      | towctrans   | truncate      | tzset      |
| ualarm       | uname       | unlink        | unsetenv   |
| usleep       | vfwprintf   | vswprintf     | vwprintf   |
| wait3        | wait4       | waitpid       | wcrtomb    |
| wcsrtombs    | wcsstr      | wctob         | wctrans    |
| wmemchr      | wmemcmp     | wmemcpy       | wmemmove   |
| wmemset      | wprintf     | wscanf        |            |



## B.4 Functions Available on *OpenVMS Alpha* Version 7.0 and Higher

Table B–4 lists functions available on *OpenVMS Alpha* Version 7.0 and higher.

**Table B–4 Functions Added in *OpenVMS Alpha* Version 7.0**

|             |             |              |              |
|-------------|-------------|--------------|--------------|
| _basename32 | _basename64 | _bsearch32   | _bsearch64   |
| _calloc32   | _calloc64   | _catgets32   | _catgets64   |
| _ctermid32  | _ctermid64  | _cuserid32   | _cuserid64   |
| _dirname32  | _dirname64  | _fgetname32  | _fgetname64  |
| _fgets32    | _fgets64    | _fgetws32    | _fgetws64    |
| _gcvt32     | _gcvt64     | _getcwd32    | _getcwd64    |
| _getname32  | _getname64  | _gets32      | _gets64      |
| _index32    | _index64    | _longname32  | _longname64  |
| _malloc32   | _malloc64   | _mbsrtowcs32 | _mbsrtowcs64 |
| _memcpy32   | _memcpy64   | _memchr32    | _memchr64    |
| _memcpy32   | _memcpy64   | _memmove32   | _memmove64   |
| _memset32   | _memset64   | _mktemp32    | _mktemp64    |
| _mmap32     | _mmap64     | _qsort32     | _qsort64     |
| _realloc32  | _realloc64  | _rindex32    | _rindex64    |
| _strcat32   | _strcat64   | _strchr32    | _strchr64    |
| _strcpy32   | _strcpy64   | _strdup32    | _strdup64    |
| _strncat32  | _strncat64  | _strncpy32   | _strncpy64   |
| _strpbrk32  | _strpbrk64  | _strptime32  | _strptime64  |
| _strrchr32  | _strrchr64  | _strsep32    | _strsep64    |
| _strstr32   | _strstr64   | _strtod32    | _strtod64    |
| _strtok32   | _strtok64   | _strtol32    | _strtol64    |
| _strtoll32  | _strtoll64  | _strtoq32    | _strtoq64    |
| _strtoul32  | _strtoul64  | _strtoull32  | _strtoull64  |
| _strtouq32  | _strtouq64  | _tmpnam32    | _tmpnam64    |
| _wscat32    | _wscat64    | _wcschr32    | _wcschr64    |
| _wscpy32    | _wscpy64    | _wcsncat32   | _wcsncat64   |
| _wcsncpy32  | _wcsncpy64  | _wcpbrk32    | _wcpbrk64    |
| _wcsrchr32  | _wcsrchr64  | _wcsrtombs32 | _wcsrtombs64 |
| _wcssstr32  | _wcssstr64  | _wcstok32    | _wcstok64    |
| _wcstol32   | _wcstol64   | _wcstoul32   | _wcstoul64   |
| _wswcs32    | _wswcs64    | _wmemchr32   | _wmemchr64   |
| _wmemcpy32  | _wmemcpy64  | _wmemmove32  | _wmemmove64  |
| _wmemset32  | _wmemset64  |              |              |

## B.5 Functions Available on *OpenVMS* Version 7.2 and Higher

Table B–5 lists functions available on *OpenVMS VAX* and *OpenVMS Alpha* Version 7.2 and higher.

**Table B–5 Functions Added in *OpenVMS* Version 7.2**

|                      |         |                                      |                            |
|----------------------|---------|--------------------------------------|----------------------------|
| asctime_r            | ctime_r | decc\$set_child_<br>standard_streams | decc\$write_eof_<br>to_mbx |
| decc\$validate_wchar | dlclose | dLError                              | dlopen                     |
| dlsym                | fcntl   | gmtime_r                             | localtime_r                |

## B.6 Functions Available on *OpenVMS* Version 7.3 and Higher

Table B–6 lists functions available on *OpenVMS VAX* and *OpenVMS Alpha* Version 7.3 and higher.

**Table B–6 Functions Added in *OpenVMS* Version 7.3**

|        |
|--------|
| fchown |
| link   |
| utime  |
| utimes |
| writev |

## B.7 Functions Available on *OpenVMS* Version 7.3-1 and Higher

Table B–7 lists functions available on *OpenVMS Alpha* Version 7.3-1 and higher.

**Table B–7 Functions Added in *OpenVMS* Version 7.3-1**

|                             |           |
|-----------------------------|-----------|
| access                      | fseeko    |
| chmod                       | ftello    |
| chown                       | ftw       |
| fstat                       | readdir_r |
|                             | stat      |
| vfscanf                     | vwscanf   |
| vscanf                      | vscanf    |
| vsscanf                     | vsscanf   |
| decc\$feature_<br>get_index |           |
| decc\$feature_<br>get_name  |           |
| decc\$feature_<br>get_value |           |

(continued on next page)

**Table B-7 (Cont.) Functions Added in *OpenVMS* Version 7.3-1**

---

|                             |
|-----------------------------|
| decc\$feature_<br>set_value |
|-----------------------------|

---



## Prototypes Duplicated to Non-Standard Headers

The various standards dictate which header file must define each of the standard functions. This is the included header file documented with each function prototype in the Reference Section of this manual.

However, many of the functions defined by the standards already existed on several operating systems and were defined in different header files. This is especially true on *OpenVMS* systems with the header files `<processes.h>`, `<unixio.h>`, and `<unixlib.h>`.

So, to provide upward compatibility for these functions, their prototypes are duplicated in both the expected header file as well as the header file defined by the standards.

Table C–1 lists these functions.

**Table C–1 Duplicated Prototypes**

| Function | Duplicated in                    | Standard says                  |
|----------|----------------------------------|--------------------------------|
| access   | <code>&lt;unixio.h&gt;</code>    | <code>&lt;unistd.h&gt;</code>  |
| alarm    | <code>&lt;signal.h&gt;</code>    | <code>&lt;unistd.h&gt;</code>  |
| bcmp     | <code>&lt;string.h&gt;</code>    | <code>&lt;strings.h&gt;</code> |
| bcopy    | <code>&lt;string.h&gt;</code>    | <code>&lt;strings.h&gt;</code> |
| bzero    | <code>&lt;string.h&gt;</code>    | <code>&lt;strings.h&gt;</code> |
| chdir    | <code>&lt;unixio.h&gt;</code>    | <code>&lt;unistd.h&gt;</code>  |
| chmod    | <code>&lt;unixio.h&gt;</code>    | <code>&lt;stat.h&gt;</code>    |
| chown    | <code>&lt;unixio.h&gt;</code>    | <code>&lt;unistd.h&gt;</code>  |
| close    | <code>&lt;unixio.h&gt;</code>    | <code>&lt;unistd.h&gt;</code>  |
| creat    | <code>&lt;unixio.h&gt;</code>    | <code>&lt;fcntl.h&gt;</code>   |
| ctermid  | <code>&lt;stdio.h&gt;</code>     | <code>&lt;unistd.h&gt;</code>  |
| cuserid  | <code>&lt;stdio.h&gt;</code>     | <code>&lt;unistd.h&gt;</code>  |
| dirname  | <code>&lt;string.h&gt;</code>    | <code>&lt;libgen.h&gt;</code>  |
| dup      | <code>&lt;unixio.h&gt;</code>    | <code>&lt;unistd.h&gt;</code>  |
| dup2     | <code>&lt;unixio.h&gt;</code>    | <code>&lt;unistd.h&gt;</code>  |
| ecvt     | <code>&lt;unixlib.h&gt;</code>   | <code>&lt;stdlib.h&gt;</code>  |
| execl    | <code>&lt;processes.h&gt;</code> | <code>&lt;unistd.h&gt;</code>  |
| execle   | <code>&lt;processes.h&gt;</code> | <code>&lt;unistd.h&gt;</code>  |

(continued on next page)

**Table C–1 (Cont.) Duplicated Prototypes**

| Function    | Duplicated in | Standard says |
|-------------|---------------|---------------|
| execlp      | <processes.h> | <unistd.h>    |
| execv       | <processes.h> | <unistd.h>    |
| execve      | <processes.h> | <unistd.h>    |
| execvp      | <processes.h> | <unistd.h>    |
| _exit       | <stdlib.h>    | <unistd.h>    |
| fcvt        | <unixlib.h>   | <stdlib.h>    |
| ffs         | <string.h>    | <strings.h>   |
| fsync       | <stdio.h>     | <unistd.h>    |
| ftime       | <time.h>      | <timeb.h>     |
| gcv         | <unixlib.h>   | <stdlib.h>    |
| getcwd      | <unixlib.h>   | <unistd.h>    |
| getegid     | <unixlib.h>   | <unistd.h>    |
| getenv      | <unixlib.h>   | <stdlib.h>    |
| geteuid     | <unixlib.h>   | <unistd.h>    |
| getgid      | <unixlib.h>   | <unistd.h>    |
| getopt      | <stdio.h>     | <unistd.h>    |
| getpid      | <unixlib.h>   | <unistd.h>    |
| getppid     | <unixlib.h>   | <unistd.h>    |
| getuid      | <unixlib.h>   | <unistd.h>    |
| index       | <string.h>    | <strings.h>   |
| isatty      | <unixio.h>    | <unistd.h>    |
| lseek       | <unixio.h>    | <unistd.h>    |
| mkdir       | <unixlib.h>   | <stat.h>      |
| mktemp      | <unixio.h>    | <stdlib.h>    |
| nice        | <stdlib.h>    | <unistd.h>    |
| open        | <unixio.h>    | <fcntl.h>     |
| pause       | <signal.h>    | <unistd.h>    |
| pipe        | <processes.h> | <unistd.h>    |
| read        | <unixio.h>    | <unistd.h>    |
| rindex      | <string.h>    | <strings.h>   |
| sbrk        | <stdlib.h>    | <unistd.h>    |
| setgid      | <unixlib.h>   | <unistd.h>    |
| setuid      | <unixlib.h>   | <unistd.h>    |
| sleep       | <signal.h>    | <unistd.h>    |
| strcasecmp  | <string.h>    | <strings.h>   |
| strncasecmp | <string.h>    | <strings.h>   |
| system      | <processes.h> | <stdlib.h>    |
| times       | <time.h>      | <times.h>     |

(continued on next page)

**Table C–1 (Cont.) Duplicated Prototypes**

| Function | Duplicated in | Standard says |
|----------|---------------|---------------|
| umask    | <stdlib.h>    | <stat.h>      |
| vfork    | <processes.h> | <unistd.h>    |
| wait     | <processes.h> | <wait.h>      |
| write    | <unixio.h>    | <unistd.h>    |





---

# Index

## A

---

abort function, 4–1, REF–3  
abs function, REF–4  
accept socket routine, A–13  
access function, REF–5  
ACCVIO  
    hardware error, 1–45  
    sigbus signal, 4–9  
    sigsegv signal, 4–10  
acos function, REF–7  
acosh function, REF–8  
addch function, REF–9  
addstr function, REF–10  
alarm function, 4–1, 4–9, REF–11  
    program example, 4–13  
Allocate memory  
    calloc function, REF–34  
    malloc function, REF–303  
    realloc function, REF–397  
\_ANSI\_C\_SOURCE macro, 1–19  
Argument list functions, 3–9 to 3–12  
Arguments  
    variable-length lists, 3–9  
ASCII  
    table of values, 3–4  
asctime function, REF–12  
asctime\_r function, REF–12  
asin function, REF–14  
asinh function, REF–15  
asm calls, 1–45  
assert function, REF–16  
AST reentrancy, 1–46, REF–91  
atan2 function, REF–18  
atan function, REF–17  
atanh function, REF–19  
atexit function, REF–20  
atof function, REF–21  
atoi function, REF–22  
atol function, REF–22  
atoll function, REF–23  
atofq function, REF–23  
Auxiliary communication routines, A–10

## B

---

Backport object library, B–1  
basename function, REF–24  
Basic communication routines, A–9  
bcmp function, REF–25  
bcopy function, REF–26  
bind socket routine, A–15  
64-bit pointer support, 1–49  
32-bit UIDs, GIDs, 1–35  
box function, REF–27  
brk function, 8–1, REF–28  
    BSD44 CURSES macro, 1–24  
bsearch function, REF–29  
btowc function, 10–10, REF–31  
bzero function, REF–32

## C

---

C\$\_LONGJMP exception, 4–11  
cabs function, REF–33  
calloc function, 8–1, REF–34, REF–44  
Carriage control  
    Fortran, 1–40  
    translation  
        by *Compaq C*, 1–39 to 1–41  
Case conversion functions, 10–9  
Case preservation in file names, xxxiii  
catclose function, 10–5, REF–35  
Categories  
    locale, 10–3  
Category  
    LC\_ALL, 10–4  
    LC\_COLLATE, 10–3  
    LC\_CTYPE, 10–3  
    LC\_MESSAGES, 10–3  
    LC\_MONETARY, 10–3  
    LC\_NUMERIC, 10–3  
    LC\_TIME, 10–3  
catgets function, 10–5, REF–36  
catopen function, 10–5, REF–39  
cbrt function, REF–42  
ceil function, REF–43  
cfree function, 8–1, REF–44

Character-classification functions, 3-4 to 3-7,  
10-9

- isalnum, REF-249
- isalpha, REF-250
- isascii, REF-252
- iscntrl, REF-254
- isdigit, REF-255
- isgraph, REF-256
- islower, REF-257
- isprint, REF-259
- ispunct, REF-260
- isspace, REF-261
- isupper, REF-262
- iswalnum, REF-263
- iswalpha, REF-264
- iswcntrl, REF-265
- iswctype, REF-266
- iswdigit, REF-268
- iswgraph, REF-269
- iswlower, REF-270
- iswprint, REF-271
- iswpunct, REF-272
- iswspace, REF-273
- iswupper, REF-274
- iswxdigit, REF-275
- isxdigit, REF-276
- program example, 3-7
- wctype, REF-661

Character-conversion functions, 3-7 to 3-9

- ecvt, REF-116
- fcvt, REF-138
- gcvt, REF-197
- toascii, REF-546
- tolower, REF-547
- \_\_tolower, REF-548
- \_\_toupper, REF-550
- \_\_toupper, REF-551
- \_\_tolower, REF-553
- \_\_toupper, REF-554
- wcswidth, REF-654
- wcwidth, REF-664

Character definition files

- location of, 10-6

Character set conversion functions

- iconv, REF-236
- iconv\_close, REF-238
- iconv\_open, REF-239

Character sets

- converting between, 10-6
- supported by *Compaq C* RTL, 10-6

Charmap file

- location, 10-6

chdir function, REF-45

Child process

- creating with `vfork`, REF-584
- executing image
  - with `exec` functions, 5-3
- implementation of, 5-2

Child process (cont'd)

- introduction to, 5-1
- program examples, 5-5
- sharing data with pipe, 5-5, REF-371
- synchronization with `wait`, 5-5

chmod function, REF-46

chown function, REF-47

C language

- I/O background, 1-36

clearerr function, REF-49

clear function, REF-48, REF-50

clearok function, REF-50

clock function, REF-51

closedir function, REF-53

close function, REF-52

close socket routine, A-17

clrattr function, REF-55

clrattr macro, 6-2

clrtoebol function, REF-56

clrtoeol function, REF-57

Codeset converter functions, 10-6

Codesets, 10-6

Communication support routines, A-10

Compaq C RTL

- 32-bit group identifiers, xxxi

- feature-enabling functions, xxix

- 2GB files, xxviii

- new functions, xxxi

- ODS-5 support, xxix

- POSIX Root, xxx, 1-28

- POSIX style identifiers, xxxi

- UNIX Style File Name Support, xxix

*Compaq C* RTL

- See Run-Time Library (RTL)

confstr function, REF-58

connect socket routine, A-18

Conversion specifications

- for I/O functions, 2-7 to 2-19

- input

- table of conversion specifiers, 2-8

- table of optional characters, 2-8

- output

- table of characters, 2-16

Converter functions

- filenaming conventions for, 10-6

copysign function, REF-60

cos function, REF-61

cosh function, REF-62

cot function, REF-63

creat function, REF-64, REF-114, REF-135,  
REF-140

crmode function, REF-70

C RTL New functions, xxxi

ctermid function, REF-71

ctime function, REF-12, REF-72

- using with `tzset` function, REF-558

- ctime\_r function, REF-72
- Cultural information
  - stored in locale, 10-7
- curscr window, 6-5
- Curses, 6-1 to 6-13
  - cursor movement, 6-10
  - getting started, 6-7 to 6-9
  - introduction to, 6-1
  - program example, 6-11
  - terminology, 6-4 to 6-7
    - curscr, 6-5
    - stdscr, 6-5
    - windows, 6-5
  - using predefined variables and constants, 6-9
- <curses.h> header file, 6-2
- Curses functions
  - box, REF-27
  - clearok, REF-50
  - delwin, REF-104
  - endwin, REF-119
  - getyx, REF-230
  - initscr, REF-243
  - leaveok, REF-284
  - longname, REF-298
  - mvcur, REF-341
  - mvwin, REF-348
  - mv[w]addch, REF-339
  - mv[w]addstr, REF-340
  - mv[w]delch, REF-342
  - mv[w]getch, REF-343
  - mv[w]getstr, REF-344
  - mv[w]inch, REF-345
  - mv[w]insch, REF-346
  - mv[w]insstr, REF-347
  - newwin, REF-349
  - [no]crmode, REF-70
  - [no]echo, REF-115
  - [no]nl, REF-353
  - [no]raw, REF-391
  - overlay, REF-364
  - overwrite, REF-365
  - scroll, REF-410
  - scrollok, REF-411
  - subwin, REF-526
  - touchwin, REF-549
  - wrapok, REF-672
  - [w]addch, REF-9
  - [w]addstr, REF-10
  - [w]clear, REF-48
  - [w]clrattr, REF-55
  - [w]clrtoebot, REF-56
  - [w]clrtoeol, REF-57
  - [w]delch, REF-101
  - [w]deleteln, REF-103
  - [w]erase, REF-121
  - [w]getch, REF-200
  - [w]getstr, REF-224
  - [w]inch, REF-241

- Curses functions (cont'd)
  - [w]insch, REF-246
  - [w]insertln, REF-247
  - [w]insstr, REF-248
  - [w]move, REF-332
  - [w]printw, REF-377
  - [w]refresh, REF-398
  - [w]scanw, REF-409
  - [w]setattr, REF-414
  - [w]standend, REF-468
  - [w]standout, REF-469
- cuserid function, REF-74

## D

---

- Date/time
  - introduction to, 11-1
- Date/time functions, 11-1 to 11-6
- Date and time functions, 10-8
- DEC/SHELL
  - See UNIX file specifications
- DECC\$ARGV\_PARSE\_STYLE feature logical, 1-28
- DECC\$CRTL\_INIT function, REF-75, REF-563
- DECC\$DEFAULT\_LRL feature logical, 1-28
- DECC\$DEFAULT\_UDF\_RECORD feature logical, 1-28
- DECC\$DETACHED\_CHILD\_PROCESS feature logical, 1-28
- DECC\$DISABLE\_POSIX\_ROOT feature logical, 1-28
- DECC\$DISABLE\_TO\_VMS\_LOGNAME\_TRANSLATION feature logical, 1-29
- DECC\$EFS\_CASE\_PRESERVE feature logical, 1-29
- DECC\$EFS\_CASE\_SPECIAL feature logical, 1-29
- DECC\$EFS\_CHARSET feature logical, 1-29
- DECC\$EFS\_FILE\_TIMESTAMPS feature logical, 1-30
- DECC\$ENABLE\_GETENV\_CACHE feature logical, 1-31
- DECC\$EXEC\_FILEATTR\_INHERITANCE feature logical, 1-31
- decc\$feature\_get\_index feature-setting routine, REF-76
- decc\$feature\_get\_name feature-setting routine, REF-78
- decc\$feature\_get\_value feature-setting routine, REF-79
- decc\$feature\_set\_value feature-setting routine, REF-80
- DECC\$FILENAME\_UNIX\_NO\_VERSION feature logical, 1-31
- DECC\$FILENAME\_UNIX\_ONLY feature logical, 1-31

DECC\$FILENAME\_UNIX\_REPORT feature  
     logical, 1-31  
 DECC\$FILE\_OWNER\_UNIX feature logical, 1-31  
 DECC\$FILE\_PERMISSION\_UNIX feature logical,  
     1-31  
 DECC\$FILE\_SHARING feature logical, 1-32  
 DECC\$FIXED\_LENGTH\_SEEK\_TO\_EOF feature  
     logical, 1-32  
 decc\$fix\_time file specification conversion  
     routine, 1-16, REF-81  
 decc\$from\_vms file specification conversion  
     routine, 1-16, REF-82  
 decc\$get\_sdc socket routine, A-20  
 DECC\$LOCALE\_CACHE\_SIZE feature logical,  
     1-32  
 DECC\$MAILBOX\_CTX\_STM feature logical, 1-32  
 decc\$match\_wild file specification conversion  
     routine, 1-16, REF-84  
 DECC\$PIPE\_BUFFER\_SIZE feature logical, 1-32  
 DECC\$POSIX\_SEEK\_STREAM\_FILE feature  
     logical, 1-32  
 DECC\$READDIR\_DROPDOTNOTYPE feature  
     logical, 1-33  
 DECC\$READDIR\_KEEPPDOTDIR feature logical,  
     1-33  
 decc\$record\_read function, REF-85  
 decc\$record\_write function, REF-86  
 DECC\$RENAME\_NO\_INHERIT feature logical,  
     1-33  
 DECC\$SELECT\_IGNORES\_INVALID\_FD feature  
     logical, 1-33  
 decc\$set\_child\_standard\_streams function,  
     REF-87  
 decc\$set\_reentrancy function, 1-47, REF-91  
 DECC\$SHR.EXE, 1-3  
 decc\$socket\_fd socket routine, A-83  
 DECC\$STDIO\_CTX\_EOL feature logical, 1-33  
 DECC\$STRTOL\_ERANGE feature logical, 1-33  
 DECC\$THREAD\_DATA\_AST\_SAFE feature  
     logical, 1-33  
 decc\$to\_vms file specification conversion routine,  
     1-16, REF-93  
 decc\$translate\_vms file specification conversion  
     routine, 1-16, REF-95  
 DECC\$TZ\_CACHE\_SIZE feature logical, 1-34  
 DECC\$UMASK feature logical, 1-34  
 DECC\$UNIX\_PATH\_BEFORE\_LOGNAME feature  
     logical, 1-34  
 DECC\$USE\_RAB64 feature logical, 1-34  
 DECC\$V62\_RECORD\_GENERATION feature  
     logical, 1-34  
 DECC\$VALIDATE\_SIGNAL\_IN\_KILL feature  
     logical, 1-34  
 decc\$validate\_wchar function, REF-97  
 decc\$write\_eof\_to\_mbx function, REF-98  
 DECC\$XPG4\_STRPTIME feature logical, 1-34

\_DECC\_SHORT\_GID\_T macro, 1-25  
 \_DECC\_V4\_SOURCE macro, 1-23  
 \_delch function, REF-101  
 delete function, REF-102, REF-399  
 deleteln function, REF-103  
 delwin function, REF-104  
 difftime function, REF-105  
 dirname function, REF-106  
 div function, REF-108  
 dlclose function, REF-109  
 dlerror function, REF-110  
 dlopen function, REF-111  
 dlsym function, REF-112  
 drand48 function, REF-113  
     using with lcong48 function, REF-281  
     using with seed48 function, REF-412  
     using with srand48 function, REF-463  
 dup2 function, REF-114, REF-140, REF-373  
 dup function, REF-114, REF-140

## E

echo function, REF-115  
 ecvt function, 3-7, REF-116  
 edata global symbol, 1-44  
 EFS, 1-18  
 end global symbol, 1-44  
 endhostent socket routine, A-21  
 endnetent socket routine, A-22  
 endprotoent socket routine, A-23  
 endpwent function, REF-118  
 endservent socket routine, A-24  
 endwin function, REF-119  
 erand48 function, REF-120  
 erase function, REF-121  
 erf function, REF-122  
 errno  
     external variable, A-4  
     relationship to h\_errno, A-7  
 <errno.h> header file, 4-2, 4-5  
 <errnode.h> header file, 4-11  
 errno variable, 4-2, 4-5, 7-4  
 Error-handling functions, 4-2  
     abort, 4-1, 4-6, 4-9, REF-3  
     error codes, 4-2  
     exit, 4-1, 5-5, REF-130  
     \_exit, 4-1, 5-5, REF-130  
     perror, 4-1, REF-370  
     strerror, 4-1, REF-483  
 ERR predefined macro, 6-10  
 etext global symbol, 1-44  
 Exact-case argv arguments, xxxiv  
 exec function, REF-372  
 exec functions  
     processing, 5-4  
 exec functions, 5-3  
     error conditions, 5-4

- execle function, REF-125
- execl function, REF-123
- execvp function, REF-126
- execve function, REF-128
- execv function, REF-127
- execvp function, REF-129
- \_exit function, 5-5
- exit, \_exit function
  - using with wait3 function, REF-603
  - using with wait4 function, REF-606
  - using with waitpid function, REF-609
- exit function, 5-5, REF-130
- \_exit function, REF-130
- \_exp function, REF-131
- Extended file specifications, xxxiii
- Extended File Specifications, 1-18

## F

- fabs function, REF-132
- fchown function, REF-133
- fclose function, REF-134, REF-159
- fcntl function, REF-135
- fcvt function, 3-7, REF-138
- fdopen function, REF-140, REF-373
- Feature logical names, 1-25
- Feature-setting routines
  - decc\$feature\_get\_index, REF-76
  - decc\$feature\_get\_name, REF-78
  - decc\$feature\_get\_value, REF-79
  - decc\$feature\_set\_value, REF-80
- Feature switches, 1-25
- Feature-test macros, 1-18
- feof function, REF-141
- ferror function, REF-142
- fflush function, REF-143
  - using with popen function, REF-374
- ffs function, REF-144
- fgetc function, REF-145
- fgetname function, REF-146
- fgetpos function, REF-147
- fgets function, REF-149
- fgetwc function, REF-151
- fgetws function, REF-152
- File
  - header, 1-1
- FILE, 2-5
- File descriptor, 2-5, 2-20
  - Compaq C defaults
  - for OpenVMS logical names, 1-18
- fileno function, REF-154
- File pointer, 2-5, 2-20
- File protection, REF-46, REF-563
- File specification conversion routines
  - decc\$fix\_time, REF-81
  - decc\$from\_vms, REF-82
  - decc\$match\_wild, REF-84
  - decc\$to\_vms, REF-93

- File specification conversion routines (cont'd)
  - decc\$translate\_vms, REF-95
- finite function, REF-155
- Fixed-length record files
  - accessing in record mode, 1-42
- Floating-point support, 1-4
- floor function, REF-156
- fmod function, REF-157
- fopen function, REF-158
- fork function, REF-584
- Format
  - specification string for input functions, 2-7
  - specification string for output functions, 2-13
- fpathconf function, REF-161
- fprintf function, REF-163
- fputc function, REF-165
- fputs function, REF-166
- fputwc function, REF-167
- fputws function, REF-168
- fp\_classf function, REF-160
- fp\_class function, REF-160
- fp\_classl function, REF-160
- fread function, REF-169
- free function, 8-1, REF-44, REF-170, REF-203
  - using with tempnam function, REF-541
- freopen function, REF-171
- frexp function, REF-172
- fscanf function, REF-174
- fseek function, 1-38, REF-176, REF-565, REF-566
- fseeko function, REF-178
- fsetpos function, REF-179
- fstat function, REF-180
- fsync function, REF-183
- ftell function, REF-184
- ftello function, REF-185
- ftime function, REF-186
- ftruncate function, REF-187
- ftw function, REF-188
- Function prototype, 1-14
- Functions
  - argument list-handling, 3-9
  - case conversion, 10-9
  - character classification, 10-9
  - character-classification, 3-4
  - character-conversion, 3-7, 3-8
  - Curses, 6-1 to 6-4
  - Date/time, 11-1 to 11-6
  - error-handling, 4-2 to 4-5
  - signal-handling, 4-5 to 4-13
  - Standard I/O, 2-1, 2-5
  - string-handling, 3-9
  - Terminal I/O, 2-19
  - Time, 11-1 to 11-6
  - UNIX I/O, 2-5
- fwait function, REF-190



fwide function, REF-191  
fwprintf function, REF-192  
fwrite function, REF-194  
fwscanf function, REF-195

## G

---

gcvf function, 3-7, REF-197  
GENCAT command, 10-5  
getc function, REF-199  
getchar function, REF-201  
getch function, REF-200  
getclock function, REF-202  
getcwd function, REF-203  
getdtablesize function, REF-204  
getegid function, REF-205  
getenv function, REF-206  
    using with putenv function, REF-380  
geteuid function, REF-208  
getgid function, REF-209  
gethostaddr socket routine, A-25  
gethostbyaddr socket routine, A-26  
gethostbyname socket routine, A-27  
gethostent socket routine, A-28  
gethostname socket routine, A-29  
getitimer function, REF-210  
getlogin function, REF-212  
getname function, REF-213, REF-372  
getnetbyaddr socket routine, A-30  
getnetbyname socket routine, A-31  
getnetent socket routine, A-32  
getopt function, REF-214  
getpagesize function, REF-217  
getpeername socket routine, A-33  
getpid function, REF-218  
getppid function, REF-219  
getprotobyname socket routine, A-34  
getprotobynumber socket routine, A-35  
getprotoent socket routine, A-36  
getpwent function, REF-220  
getpwnam function, REF-221  
getpwuid function, REF-222  
getservbyname socket routine, A-37  
getservbyport socket routine, A-38  
getservent socket routine, A-39  
gets function, REF-149, REF-223  
getsockname socket routine, A-40  
getsockopt socket routine, A-41  
getstr function, REF-224  
gettimeofday function, REF-225  
getuid function, REF-226  
getwc function, REF-228  
getwchar function, REF-229  
getw function, REF-227  
getyx function, REF-230  
2-gigabyte files, 1-25

gmtime function, REF-231  
gmtime\_r function, REF-231  
gsignal function, 4-5, 4-9, REF-233

## H

---

Header files, 1-1, 1-15  
    displaying on Alpha systems, 1-1  
    displaying on VAX systems, 1-1  
herror socket routine, A-43  
hostalias socket routine, A-45  
hstrerror socket routine, A-44  
htonl socket routine, A-46  
htons socket routine, A-47  
hypot function, REF-235  
h\_errno  
    external variable, A-6  
    relationship to errno, A-7  
    support routines, A-10

## I

---

iconv function, 10-6, REF-236  
iconv\_close function, 10-6, REF-238  
iconv\_open function, 10-6, REF-239  
IMAGELIB.OLB, 1-3  
inch function, REF-241  
index function, REF-242  
inet\_addr socket routine, A-48  
inet\_lnaof socket routine, A-49  
inet\_makeaddr socket routine, A-50  
inet\_netof socket routine, A-51  
inet\_network socket routine, A-52  
inet\_ntoa socket routine, A-53  
initscr function, REF-243  
initstate function, REF-244  
    using with setstate function, REF-428  
Input and output (I/O), 1-35 to 1-41  
    conversion specifications, 2-7 to 2-19  
    format specification string, 2-7, 2-13  
    OpenVMS system services, 1-35  
    record access  
        in Compaq C, 1-39  
    Record Management Services (RMS), 1-35  
    Standard, 1-35  
    stream access  
        in Compaq C, 1-39  
    UNIX, 1-35  
    Wide-character, 2-6  
insch function, REF-246  
insertln function, REF-247  
insstr function, REF-248  
insstr macro, 6-2  
Internationalization support, 10-1  
International software  
    description of, 10-2

## Internet

- application programs, A-1
- protocols, A-2

## Interprocess communication, 5-1

- ioctl socket routine, A-54
- IPv6 socket routines, A-12
- isalnum function, REF-249
- isalpha function, REF-250
- isapipe function, REF-251
- isascii function, REF-252
- isatty function, REF-253
- isctrl function, REF-254
- isdigit function, REF-255
- isgraph function, REF-256
- islower function, REF-257
- isnan function, REF-258
- isprint function, REF-259
- ispunct function, REF-260
- isspace function, REF-261
- isupper function, REF-262
- iswalnum function, REF-263
- iswalpha function, REF-264
- iswcntrl function, REF-265
- iswctype function, REF-266
- iswdigit function, REF-268
- iswgraph function, REF-269
- iswlower function, REF-270
- iswprint function, REF-271
- iswpunct function, REF-272
- iswspace function, REF-273
- iswupper function, REF-274
- iswxdigit function, REF-275
- isxdigit function, REF-276
- itimerval structure, REF-210, REF-419

## J

- j0 function, REF-277
- j1 function, REF-277
- jn function, REF-277
- rand48 function, REF-278

## K

- kill function, REF-279

## L

- labs function, REF-280
- LANG logical name, 10-5
- LARGEFILE macro, 1-25
- Large files, 1-25
- lcong48 function, REF-281
  - using with drand48 function, REF-113
  - using with lrand48 function, REF-299
  - using with mrand48 function, REF-335

## LC\_ALL category, 10-4

- LC\_ALL logical name, 10-5
- LC\_CTYPE category, 10-9
- LC\_NUMERIC logical name, 10-5
- ldexp function, REF-282
- ldiv function, REF-283
- leaveok function, REF-284
- lgamma function, REF-285
- LIB\$ESTABLISH function, 4-11, REF-577
- LIB\$SIGNAL, 4-9

## Libraries

- Compaq C RTL object-module linking order, 1-8

## Library

- main function, 1-2

## Linker

- search libraries, 1-2

## link function, REF-286

## Linking

- with RTL object libraries, 1-4

## Linking with the C RTL, 1-3 to 1-14

## listen socket routine, A-55

## List-handling functions

- va\_arg, REF-579
- va\_count macro, REF-580
- va\_end, REF-581
- va\_start, REF-582
- va\_start\_1, REF-582

## llabs function, REF-385

## lldiv function, REF-386

## LNK\$LIBRARY logical name, 1-3, 1-8

## Locale

- categories, 10-3
- description of, 10-3
- extracting information from, 10-8

## localeconv function, 10-8, REF-287

## Locale support functions

- localeconv, REF-287
- nl\_langinfo, REF-354
- setlocale, REF-423

## localtime function, REF-291

- using with tzset function, REF-558

## localtime\_r function, REF-291

## log10 function, REF-293

## loglp function, REF-294

## logb function, REF-295

## log function, REF-293

## Logical name

- for default locale, 10-5
- for default locale categories, 10-5
- for international environment, 10-5
- for locale directory, 10-3
- for system default locale, 10-5
- LANG, 10-5
- LC\_ALL, 10-5
- LC\_NUMERIC, 10-5
- SYS\$I18N\_LOCALE, 10-3
- SYS\$LANG, 10-5

Logical name (cont'd)

    SYS\$LC\_ALL, 10-5

Logical names

    feature, 1-25

Long file names

    arguments to C RTL functions, xxxiii

longjmp function, 4-11, REF-296, REF-577,  
    REF-584

longjmp member

    using with ftw function, REF-189

longname function, REF-298

lrand48 function, REF-299

    using with lcong48 function, REF-281

    using with seed48 function, REF-412

    using with srand48 function, REF-463

lseek function, 1-38, REF-300

lstat function

    using with ftw function, REF-188

lwait function, REF-302

## M

---

Macros

    feature-test, 1-18

main function

    using with wait3 function, REF-603

    using with wait4 function, REF-606

    using with waitpid function, REF-609

Main function, 1-2, 4-10

    main\_program option, 1-2

malloc function, 8-1, REF-44, REF-303

    using with ftw function, REF-189

    using with putenv function, REF-380

Math functions, 7-1 to 7-6

    abs, REF-4

    acos, REF-7

    acosh, REF-8

    asin, REF-14

    asinh, REF-15

    atan, REF-17

    atan2, REF-18

    atanh, REF-19

    cabs, REF-33

    cbrt, REF-42

    ceil, REF-43

    copysign, REF-60

    cos, REF-61

    cosh, REF-62

    cot, REF-63

    div, REF-108

    erf, REF-122

    errno values, 7-1

    exp, REF-131

    fabs, REF-132

    finite, REF-155

    floor, REF-156

    fp\_class, REF-160

    fp\_classf, REF-160

Math functions (cont'd)

    fp\_classl, REF-160

    frexp, REF-172

    hypot, REF-235

    isnan, REF-258

    j0, REF-277

    j1, REF-277

    jn, REF-277

    labs, REF-280

    ldexp, REF-282

    ldiv, REF-283

    lgamma, REF-285

    llabs, REF-385

    lldiv, REF-386

    log, REF-293

    log10, REF-293

    loglp, REF-294

    logb, REF-295

    modf, REF-331

    nextafter, REF-350

    nint, REF-352

    pow, REF-375

    qabs, REF-385

    qdiv, REF-386

    rand, REF-389

    rint, REF-404

    scalb, REF-407

    sin, REF-456

    sinh, REF-457

    sqrt, REF-461

    srand, REF-462

    tan, REF-538

    tanh, REF-539

    trunc, REF-555

    unordered, REF-567

    y0, REF-677

    y1, REF-677

    yn, REF-677

mblen function, REF-305

mbrlen function, REF-306

mbrtowc function, 3-7, 10-10, REF-307

mbsinit function, REF-311

mbsrtowcs function, 3-7, 10-10, REF-312

mbstate\_t, 10-10, REF-306, REF-307,  
    REF-311, REF-312, REF-612, REF-636

mbstowcs function, 10-10, REF-309

mbtowc function, 3-7, 10-10, REF-310

memcpy function, REF-314

memchr function, REF-315

memcmp function, REF-316

memcpy function, REF-317

memmove function, REF-318

Memory allocation

    introduction to, 8-1

    program examples, 8-2

Memory allocation functions

    brk, REF-28

    calloc, REF-34



Memory allocation functions (cont'd)

- cfree, REF-44
- free, REF-170
- malloc, REF-303
- realloc, REF-397
- sbrk, REF-406

Memory reallocation, REF-170

memset function, REF-319

Message catalog, 10-3

- creating, 10-5

Messaging functions

- catclose, REF-35
- catgets, REF-36
- catopen, REF-39

mkdir function, REF-320

mkstemp function, REF-323

mktemp function, REF-324

mktime function, REF-325

- using with tzset function, REF-558

mmap function, REF-326

modf function, REF-331

Modules

- Compaq C* RTL object linking order, 1-8

Monetary formatting function

- strfmon, REF-485

Monetary function, 10-8

move function, REF-332

mprotect function, REF-333

rand48 function, REF-335

- using with lcong48 function, REF-281
- using with seed48 function, REF-412
- using with srand48 function, REF-463

msync function, REF-336

Multibyte character

- conversion to wide character, 10-10

Multibyte character support

- btowc, REF-31
- mblen, REF-305
- mbrlen, REF-306
- mbrtowc, REF-307
- mbsinit, REF-311
- mbtowc, REF-310
- wcrtomb, REF-612
- wctob, REF-658
- wctomb, REF-659

Multibyte string support

- mbsrtowcs, REF-312
- mbstowcs, REF-309
- wcsrtombs, REF-636
- wcstombs, REF-648

MULTITHREAD reentrancy, 1-46, REF-91

Multithread Restrictions, 1-48

munmap function, REF-338

mvaddch function, REF-339

mvaddstr function, REF-340

mvcur function, REF-341

mvdelch function, REF-342

mvgetch function, REF-343

mvgetstr function, REF-344

mvinch function, REF-345

mvinsch function, REF-346

mvinsstr function, REF-347

mvinsstr macro, 6-2

mvwaddch function, REF-339

mvwaddstr function, REF-340

mvwdelch function, REF-342

mvwgetch function, REF-343

mvwgetstr function, REF-344

mvwinch function, REF-345

mvwin function, REF-348

mvwinsch function, REF-346

mvwinsstr function, REF-347

mvwinsstr macro, 6-2

## N

Nested directory limitation lifted, xxxiii

New functions, xxxv

newwin function, REF-349

nextafter function, REF-350

nice function, REF-351

nint function, REF-352

nl function, REF-353

nl\_langinfo function, 10-8, REF-354

nochrmode function, REF-70

noecho function, REF-115

NONE reentrancy, REF-91

nonl function, REF-353

noraw function, REF-391

nrnd48 function, REF-358

ntohl socket routine, A-56

ntohs socket routine, A-57

## O

Object libraries

- RTL, 1-4

Object library

- backport, B-1
- DECC\$CRTL, B-1
- VAXCRTL.OLB, 1-4
- VAXCRTLD.OLB, 1-4
- VAXCRTLDX.OLB, 1-4
- VAXCRTLT.OLB, 1-4
- VAXCRTLTIX.OLB, 1-4
- VAXCRTLX.OLB, 1-4

Object module

- Compaq C* RTL linking order, 1-8

Occlusion, 6-4

ODS-5 volumes, 1-18

opendir function, REF-362

- using with readdir function, REF-395
- using with rewinddir function, REF-402

open function, REF-114, REF-135, REF-140, REF-359  
*OpenVMS* system services  
    in *Compaq C* programs, 1-35  
*OpenVMS* versions, B-1  
Operating system version dependency, B-1  
overlay function, REF-27, REF-364  
overwrite function, REF-27, REF-365

## P

---

passwd structure, REF-220, REF-221, REF-222  
pathconf function, REF-366  
pause function, REF-368  
pclose function, REF-369  
    using with popen function, REF-374  
perror function, 4-5, REF-370  
pipe function, REF-114, REF-135, REF-140, REF-371  
Pointers  
    64-bit support, 1-49  
popen function, REF-374  
Portability concerns, 1-36  
    arguments to mkdir, REF-320  
    \_exit function, REF-130  
    gsignal function, REF-233  
    longname function, REF-298  
    memory deallocation, REF-44  
    mcur function, 6-10  
    mv[w]insstr functions, REF-347  
    [no]nl functions, REF-353  
    radix conversion specifiers, 2-11  
    raise function, REF-388  
    socket routines, A-1  
    specific  
        list of, 1-44 to 1-46  
    ssignal function, REF-467  
    ttyname function, REF-557  
    UNIX file specifications, 1-16  
        ambiguity of, 1-17  
    variable-length argument lists, 3-9  
    va\_start\_1 macro, REF-582  
    vfork versus fork function, REF-584  
    [w]clrattrib functions, REF-55  
    [w]insstr functions, REF-248  
    [w]setattr functions, REF-414  
POSIX Root Support, 1-28  
POSIX style identifiers, 1-35  
POSIX style root, xxx  
    \_POSIX\_C\_SOURCE macro, 1-19  
    \_POSIX\_EXIT macro, 1-23  
pow function, REF-375  
Predefined macro  
    ERR, 6-10  
Predefined variables and constants, 6-9  
printf function, REF-376

printf function, REF-377  
Process permanent files, 2-19  
Protocols  
    Internet, A-2  
putc function, REF-378  
putchar function, REF-379  
putenv function, REF-380  
puts function, REF-381  
putwc function, REF-383  
putwchar function, REF-384  
putw function, REF-382

## Q

---

qabs function, REF-385  
qdiv function, REF-386  
qsort function, REF-387  
Quotas  
    affecting RTL, 5-1, 5-2, 5-4

## R

---

raise function, 4-5, 4-9, REF-279, REF-388  
rand function, REF-389  
random function, REF-390  
raw function, REF-391  
readdir function, REF-395  
    using with closedir function, REF-53  
readdir\_r function, REF-395  
Reader's comments, xxv  
read function, REF-393  
read socket routine, A-58  
realloc function, 8-1, REF-397  
Record  
    access by *Compaq C*, 1-39  
    I/O  
        *Compaq C* handling of, 1-40  
Record files  
    accessing in record mode, 1-39  
    accessing in stream mode, 1-39  
Record Management Services (RMS)  
    accessing files, 1-38  
    file organization, 1-37  
    in *Compaq C* programs, 1-35  
    overview of, 1-37 to 1-41  
    record access  
        in *Compaq C*, 1-39  
    record formats, 1-38  
    stream access  
        in *Compaq C*, 1-39  
recvfrom socket routine, A-61  
recvmsg socket routine, A-63  
recv socket routine, A-59  
Reentrancy, 1-46, REF-91  
    AST, 1-46, REF-91  
    MULTITHREAD, 1-46, REF-91  
    NONE, REF-91  
    Restrictions, 1-48

## Reentrancy (cont'd)

TOLERANT, 1-46, REF-91

## Reentrancy function

decc\$set reentrancy, 1-47, REF-91

/REENTRANCY qualifier, 1-47

refresh function, REF-50, REF-398

remove function, REF-102, REF-399

rename function, REF-400

## rewinddir function

using with readdir function, REF-395

rewind function, REF-401

rindex function, REF-403

rint function, REF-404

rmdir function, REF-405

## RMS

file attributes, 2-4

## Routines

socket, A-1 to A-106

## Run-Time Library (RTL)

as shared images, 1-2

Curses functions and macros, 6-1

Date/time functions, 11-1

header files, 1-15

I/O, 1-35 to 1-41

Compaq C handling of, 1-38 to 1-41

interpreting syntax, 1-14

introduction to, 1-1 to 1-56

linking against RTL object libraries, 1-4, 1-8

linking against RTL shareable image, 1-3

linking options explained, 1-3 to 1-14

portability concerns, 1-36

preprocessor directives, 1-15

specific portability concerns, 1-44 to 1-46

stream I/O, 1-39

# S

sbrk function, 8-1, REF-406

scalb function, REF-407

scanf function, REF-408

scanw function, REF-409

## Screen management

Curses

See Curses

scroll function, REF-410

scrollok function, REF-411

## Security/impersonation functions

getegid, REF-205

geteuid, REF-208

getgid, REF-209

getuid, REF-226

setgid, REF-417

setuid, REF-429

seed48 function, REF-412

using with drand48 function, REF-113

using with lcong48 function, REF-281

using with lrand48 function, REF-299

using with mrand48 function, REF-335

seekdir function, REF-413

select socket routine, A-65

sendmsg socket routine, A-70

send socket routine, A-68

sendto socket routine, A-72

setattr function, REF-414

setattr macro, 6-2

setbuf function, REF-415

setenv function, REF-416

setgid function, REF-417

sethostent socket routine, A-74

setitimer function, REF-419

using with ualarm function, REF-562

setjmp function, 4-11, REF-296, REF-421,

REF-577, REF-584

setlocale function, 10-4, REF-423

setnetent socket routine, A-75

setprotoent socket routine, A-76

setpwent function, REF-427

setservent socket routine, A-77

setsockopt socket routine, A-78

setstate function, REF-428

using with initState function, REF-244

setuid function, REF-429

setvbuf function, REF-431

using with popen function, REF-374

Shareable image, 1-3

Shared access, xxxiv

## Shared image

Compaq C RTL, 1-2

shutdown socket routine, A-80

sigaction function, REF-433

sigaction structure, REF-433

sigaddset function, REF-436

sigblock function, 4-6, REF-437, REF-445

sigdelset function, REF-438

sigemptyset function, REF-439

sigfillset function, REF-440

sigismember function, REF-441

siglongjmp function, REF-442

sigmask function, REF-443

<signal.h> header file, 4-6

signal function, 4-6, 4-10, REF-233, REF-388,

REF-444, REF-467

## Signal handler

calling interface, 4-7

## Signal-handling functions, 4-5

alarm, REF-11

gsignal, REF-233

kill, REF-279

longjmp, REF-296

OpenVMS exceptions, 4-9

pause, REF-368

program examples, 4-13

raise, REF-388

setjmp, REF-421

sigaction, REF-433

sigaddset, REF-436

## Signal-handling functions (cont'd)

- sigblock, REF-437
- sigdelset, REF-438
- sigemptyset, REF-439
- sigfillset, REF-440
- sigismember, REF-441
- siglongjmp, REF-442
- sigmask, REF-443
- signal, REF-444
- sigpause, REF-445
- sigpending, REF-446
- sigprocmask, REF-447
- sigsetjmp, REF-449
- sigsetmask, REF-451
- sigstack, REF-452
- sigsuspend, REF-454
- sigvec, REF-455
- sleep, REF-458
- ssignal, REF-467
- UNIX signals, 4-6
- VAXC\$ESTABLISH, REF-577

## Signals, 4-5

- sigpause function, REF-445
- sigpending function, REF-446
- sigprocmask function, REF-447
- sigsetjmp function, REF-449
- sigsetmask function, 4-6, REF-451
- sigstack function, REF-452
- sigsuspend function, REF-454
- sigvec function, 4-6, 4-10, REF-233, REF-388, REF-455

- sin function, REF-456

- sinh function, REF-457

- sleep function, REF-458

- \_SOCKADDR\_LEN macro, 1-24

## Socket routines

- accept, A-13
- auxiliary communication routines, A-10
- basic communication routines, A-9
- bind, A-15
- close, A-17
- communication support routines, A-10
- Compaq C structures, A-2
- connect, A-18
- decc\$get\_sdc, A-20
- decc\$socket\_fd, A-83
- endhostent, A-21
- endnetent, A-22
- endprotoent, A-23
- endservent, A-24
- gethostaddr, A-25
- gethostbyaddr, A-26
- gethostbyname, A-27
- gethostent, A-28
- gethostname, A-29
- getnetbyaddr, A-30
- getnetbyname, A-31
- getnetent, A-32

## Socket routines (cont'd)

- getpeername, A-33
- getprotobyname, A-34
- getprotobynumber, A-35
- getprotoent, A-36
- getservbyname, A-37
- getservbyport, A-38
- getservent, A-39
- getsockname, A-40
- getsockopt, A-41
- herror, A-43
- hostalias, A-45
- hstrerror, A-44
- htonl, A-46
- htons, A-47
- h\_errno support routines, A-10
- inet\_addr, A-48
- inet\_lnaof, A-49
- inet\_makeaddr, A-50
- inet\_netof, A-51
- inet\_network, A-52
- inet\_ntoa, A-53
- introduction, A-1
- ioctl, A-54
- IPv6 routines, A-12
- listen, A-55
- ntohl, A-56
- ntohs, A-57
- porting considerations, A-1
- programming examples, A-86
- read, A-58
- recv, A-59
- recvfrom, A-61
- recvmsg, A-63
- select, A-65
- send, A-68
- sendmsg, A-70
- sendto, A-72
- sethostent, A-74
- setnetent, A-75
- setprotoent, A-76
- setservent, A-77
- setsockopt, A-78
- shutdown, A-80
- socket, A-81
- vaxc\$get\_sdc, A-84
- write, A-85

- socket socket routine, A-81

## Specification delimiters

- OpenVMS* and UNIX, 1-16

- sprintf function, REF-459

- sqr function, REF-461

- srand48 function, REF-463

- using with drand48 function, REF-113

- using with lcong48 function, REF-281

- using with lrand48 function, REF-299

- using with mrand48 function, REF-335

- rand function, REF-462
- random function, REF-464
  - using with random function, REF-390
- sscanf function, REF-465
- ssignal function, 4-6, REF-233, REF-388, REF-467
- Standard header file, 1-1
- Standard I/O, 1-35
  - file pointers, 2-5
  - introduction to, 2-1
  - program example, 2-22
  - wide character, 2-23
- Standard I/O functions
  - clearerr, REF-49
  - delete, REF-102, REF-399
  - dlclose, REF-109
  - dLError, REF-110
  - dlopen, REF-111
  - dlsym, REF-112
  - fclose, REF-134
  - fdopen, REF-140
  - feof, REF-141
  - ferror, REF-142
  - fflush, REF-143
  - fgetc, REF-145
  - fgetname, REF-146
  - fgets, REF-149
  - fgetwc, REF-151
  - fgetws, REF-152
  - fopen, REF-158
  - fprintf, REF-163
  - fputc, REF-165
  - fputs, REF-166
  - fputwc, REF-167
  - fputws, REF-168
  - fread, REF-169
  - freopen, REF-171
  - fscanf, REF-174
  - fseek, REF-176
  - fseeko, REF-178
  - ftell, REF-184
  - ftello, REF-185
  - fwrite, REF-194
  - getc, REF-199
  - getw, REF-227
  - getwc, REF-228
  - mktemp, REF-324
  - putc, REF-378
  - putw, REF-382
  - putwc, REF-383
  - rewind, REF-401
  - setbuf, REF-415
  - setvbuf, REF-431
  - sprintf, REF-459
  - sscanf, REF-465
  - tmpfile, REF-544
  - tmpnam, REF-545
  - ungetc, REF-565

- Standard I/O functions (cont'd)
  - ungetwc, REF-566
- Standards
  - listed, 1-19
- standend function, REF-468
- standout function, REF-469
- stat function, REF-470
- stat structure
  - using with ftw function, REF-188
  - \_STDC\_VERSION\_ macro, 1-19
- stderr, 2-20, REF-143, REF-171, REF-370, REF-372
- stdin, 2-20, REF-171, REF-372, REF-408
- <stdio.h> header file, 2-20
- stdout, 2-20, REF-171, REF-372, REF-376, REF-379, REF-381, REF-384
- stdscr window, 6-5
- strcascmp function, REF-473
- strcat function, REF-474
- strchr function, REF-315, REF-476
- strcmp function, REF-316, REF-478
- strcmpn function, 1-45
- strcoll function, 10-11, REF-479
- strcpy function, REF-317, REF-480
- strcpyn function, 1-45
- strcspn function, REF-481
- strdup function, REF-482
- Stream
  - access by *Compaq C*, 1-39
  - files, 2-1
- strerror function, 4-5, REF-483
- strfmon function, 10-8, REF-485
- strftime function, 10-8, REF-489
  - using with tzset function, REF-558
- String comparison functions
  - multipass collation, 10-11
  - wcscoll, REF-618
- String-handling functions, 3-9 to 3-10
  - atof, REF-21
  - atoi, REF-22
  - atol, REF-22
  - atoll, REF-23
  - atof, REF-23
  - basename, REF-24
  - bcmp, REF-25
  - bcopy, REF-26
  - bzero, REF-32
  - dirname, REF-106
  - ffs, REF-144
  - index, REF-242
  - memchr, REF-315
  - memcmp, REF-316
  - memcpy, REF-317
  - memmove, REF-318
  - memset, REF-319
  - program examples, 3-10
  - rindex, REF-403
  - strcascmp, REF-473



## String-handling functions (cont'd)

- strcat, REF-474
- strchr, REF-476
- strcmp, REF-478
- strcoll, REF-479
- strcpy, REF-480
- strcspn, REF-481
- strdup, REF-482
- strlen, REF-495
- strncasecmp, REF-496
- strncat, REF-497
- strncmp, REF-498
- strncpy, REF-500
- strnlen, REF-501
- strpbrk, REF-502
- strrchr, REF-508
- strsep, REF-509
- strspn, REF-510
- strtok, REF-515
- strtol, REF-517
- strtoll, REF-519
- strtoq, REF-519
- strtoul, REF-521
- strtoull, REF-522
- strtouq, REF-522
- strxfrm, REF-523
- swab, REF-527
- wscat, REF-613
- wchr, REF-615
- wscmp, REF-617
- wscoll, REF-618
- wscopy, REF-619
- wscspn, REF-620
- wcslen, REF-628
- wcsncat, REF-629
- wcsncmp, REF-631
- wcsncpy, REF-632
- wcspbrk, REF-633
- wcsrchr, REF-634
- wcsspn, REF-638
- wcstok, REF-643
- wcstol, REF-646
- wcstoul, REF-649
- wcswcs, REF-652
- wcsxfrm, REF-655
- strlen function, REF-495
- strncasecmp function, REF-496
- strncat function, REF-497
- strncmp function, REF-498
- strncpy function, REF-500
- strnlen function, REF-501
- strpbrk function, REF-502
- strptime function, xxxii, 10-8, REF-503
- strrchr function, REF-508
- strsep function, REF-509
- strspn function, REF-510

- strstr function, REF-511
- strtod function, 10-8, REF-21, REF-513
- strtok function, REF-515
- strtol function, REF-22, REF-23, REF-517
- strtoll function, REF-519
- strtoq function, REF-519
- strtoul function, REF-521
- strtoull function, REF-522
- strtouq function, REF-522
- Structures
  - use with socket routines, A-2
- strxfrm function, 10-11, REF-523
- Subprocess, 5-1 to 5-11
  - executing image
    - with exec functions, 5-3
  - implementation of, 5-2
  - introduction to, 5-1
  - program examples, 5-5 to 5-11
  - sharing data with pipe, 5-5, REF-371
  - synchronization with wait, 5-5
- Subprocess functions
  - decc\$set\_child\_standard\_streams, REF-87
  - decc\$validate wchar, REF-97
  - decc\$write\_eof\_to\_mbx, REF-98
  - execl, REF-123
  - execle, REF-125
  - execlp, REF-126
  - execv, REF-127
  - execve, REF-128
  - execvp, REF-129
  - pipe, REF-371
  - vfork, REF-584
  - wait, REF-601
- subwin function, REF-526
- swab function, REF-527
- swprintf function, REF-528
- swscanf function, REF-529
- Synchronizing processes, 5-5
- Syntax
  - of Compaq C RTL functions, 1-14
- SYS\$ERROR, 2-20
- SYS\$I18N\_LOCALE logical name, 10-3
- SYS\$INPUT, 2-20
- SYS\$LANG logical name, 10-5
- SYS\$LC\_ALL logical name, 10-5
- SYS\$OUTPUT, 2-20
- SYS\$POSIX\_ROOT, xxx, 1-28
- SYS\$WAKE, REF-11
- sysconf function, REF-530
- system function, REF-536
- System functions, 9-1 to 9-5
  - asctime, REF-12
  - asctime\_r, REF-12
  - assert, REF-16
  - atexit, REF-20
  - bsearch, REF-29
  - chdir, REF-45

## System functions (cont'd)

- chmod, REF-46
- chown, REF-47
- clock, REF-51
- ctermid, REF-71
- ctime, REF-72
- ctime\_r, REF-72
- cuserid, REF-74
- difftime, REF-105
- fchown, REF-133
- fmod, REF-157
- ftime, REF-186
- getcwd, REF-203
- getenv, REF-206
- getpid, REF-218
- getppid, REF-219
- gmtime, REF-231
- gmtime\_r, REF-231
- introduction to, 9-3
- localtime, REF-291
- localtime\_r, REF-291
- memset, REF-319
- mkdir, REF-320
- nice, REF-351
- program examples, 9-3
- qsort, REF-387
- remove, REF-102, REF-399
- rename, REF-400
- setbuf, REF-415
- setvbuf, REF-431
- strtod, REF-513
- strtok, REF-515
- system, REF-536
- time, REF-542
- times, REF-543
- umask, REF-563
- utime, REF-568
- utimes, REF-571
- vfprintf, REF-586
- vfscanf, REF-587
- vprintf, REF-592
- vscanf, REF-593
- vsprintf, REF-594
- vsscanf, REF-595
- wctod, REF-641
- wcstok, REF-643
- wrotev, REF-674

## T

- tan function, REF-538
- tanh function, REF-539
- TCP/IP Services for OpenVMS*, A-1
- tellmdir function, REF-540
- tempnam function, REF-541
- Terminal I/O
  - program examples, 2-20 to 2-25

## Terminal I/O functions

- getchar, REF-201
- gets, REF-223
- getwchar, REF-229
- printf, REF-376
- putchar, REF-379
- puts, REF-381
- putwchar, REF-384
- scanf, REF-408

## Time

- introduction to, 11-1

- time function, REF-542

- Time functions, 11-1 to 11-6

## Time-related functions

- asctime, REF-12
- asctime\_r, REF-12
- clock, REF-51
- ctime, REF-72
- ctime\_r, REF-72
- decc\$fix\_time, REF-81
- difftime, REF-105
- ftime, REF-186
- getclock, REF-202
- getitimer, REF-210
- gettimeofday, REF-225
- gmtime, REF-231
- gmtime\_r, REF-231
- localtime\_r, REF-291
- mktime, REF-325
- setitimer, REF-419
- strftime, REF-489
- strptime, REF-503
- time, REF-542
- times, REF-543
- tzset, REF-558
- ualarm, REF-562
- usleep, REF-575
- utime, REF-568
- utimes, REF-571
- wcsftime, REF-622
- times function, REF-543
- timespec structure, REF-202
- Time-zone cache, REF-561
- tmpfile function, REF-544
- tmpnam function, REF-545
- toascii function, 3-7, REF-546
- TOLERANT reentrancy, 1-46, REF-91
- \_tolower macro, 3-7
- \_tolower function, REF-548
- tolower function, 3-7, REF-547
- touchwin function, REF-549
- \_toupper macro, 3-7
- \_toupper function, REF-551
- toupper function, 3-7, REF-550
- towctrans function, 3-7, 10-9, REF-552
- towlower function, 10-9, REF-553

tolower function, 10–9, REF–554  
truncate function, REF–556  
trunc function, REF–555  
ttyname function, REF–557  
tzset function, REF–558

## U

---

ualarm function, REF–562  
umask function, REF–563  
umask value, 5–4  
uname function, REF–564  
ungetc function, REF–565  
ungetwc function, REF–566  
UNIX  
    file specification conversion functions, 1–16  
    file specifications, 1–16 to 1–18  
        alternate translation, 1–17  
        compared to *OpenVMS*, 1–16  
    Run-Time Library, 1–16  
    use with *Compaq C* RTL, 1–16 to 1–18  
UNIX file specifications  
    alternate translation, xxxiv  
UNIX I/O, 1–35  
    file descriptors, 2–5  
    functions  
        program example, 2–24  
UNIX I/O functions  
    close, REF–52  
    creat, REF–64  
    dup, REF–114  
    dup2, REF–114  
    fcntl, REF–135  
    fileno, REF–154  
    fstat, REF–180  
    getname, REF–213  
    getopt, REF–214  
    isapipe, REF–251  
    isatty, REF–253  
    lseek, REF–300  
    open, REF–359  
    read, REF–393  
    stat, REF–470  
    ttyname, REF–557  
    write, REF–673  
UNIX style root, xxx, 1–28  
unordered function, REF–567  
unsetenv function, REF–574  
    USE\_LONG\_GID\_T macro, 1–25  
usleep function, REF–575  
utime function, REF–568  
utimes function, REF–571

## V

---

<varargs.h> header file, 3–9  
Variable-length argument lists, 3–9  
Variable-length record files  
    accessing in record mode, 1–41  
VAXC\$CTRL\_INIT function, 4–10, REF–563,  
    REF–576  
vaxc\$errno external variable, 4–5  
VAXC\$ESTABLISH function, 4–11, REF–296,  
    REF–421, REF–577  
VAXC\$EXECMBX logical name, 5–4  
vaxc\$get\_sdc socket routine, A–84  
va\_arg function, REF–579  
va\_count macro, REF–580  
va\_end function, REF–581  
va\_start macro, REF–582  
va\_start\_1 macro, REF–582  
Version-dependency of Compaq C RTL routines,  
    B–1  
vfork function, REF–372, REF–584  
vfprintf function, REF–586  
vfscanf function, REF–587  
vfwprintf function, REF–589  
vfwscanf function, REF–591  
    \_VMS\_CURSES macro, 1–24  
    \_VMS\_V6\_SOURCE macro, 1–23  
    \_VMS\_VER macro, 1–22  
    \_VMS\_VER\_OVERRIDE macro, 1–23  
vprintf function, REF–592  
vscanf function, REF–593  
vsprintf function, REF–594  
vsscanf function, REF–595  
vswprintf function, REF–596  
vswscanf function, REF–598  
vwprintf function, REF–599  
vwscanf function, REF–600

## W

---

waddch function, REF–9  
waddstr function, REF–10  
wait3 function, REF–602  
wait4 function, REF–605  
wait function, REF–601  
    using with waitpid function, REF–608  
waitpid function, REF–608  
wclear function, REF–48  
wclratr function, 6–2, REF–55  
wclrtoeb function, REF–56  
wclrtoeol function, REF–57  
wrtomb function, 3–7, 10–10, REF–612  
wscat function, REF–613  
wcschr function, REF–615  
wcscmp function, REF–617



- wcscoll function, 10–11, REF–618
- wscspy function, REF–619
- wscscspn function, REF–620
- wcsftime function, 10–8, REF–622
- wcslen function, REF–628
- wcsncat function, REF–629
- wcsncmp function, REF–631
- wcsncpy function, REF–632
- wcspbrk function, REF–633
- wcsrchr function, REF–634
- wcsrtombs function, 3–7, 10–10, REF–636
- wcsspn function, REF–638
- wcsstr function, REF–640
- wcstod function, 10–8, REF–641
- wcstok function, REF–643
- wcstol function, REF–646
- wcstombs function, 10–10, REF–648
- wcstoul function, REF–649
- wcswcs function, REF–652
- wcswidth function, REF–654
- wcsxfrm function, 10–11, REF–655
- wctob function, 10–10, REF–658
- wctomb function, 10–10, REF–659
- wctrans function, 3–7, 10–9, REF–660
- wctype function, REF–661
- wcwidth function, REF–664
- wdelch function, REF–101
- wdeleteln function, REF–103
- werase function, REF–121
- wgetch function, REF–115, REF–200
- wgetstr function, REF–115, REF–224
- Wide character
  - collating functions, 10–11
  - conversion to multibyte, 10–10
  - data type, 10–9
  - functions, 10–9
  - I/O functions, 10–10
- Wide-character functions
  - btowc, REF–31
  - fgetwc, REF–151
  - fgetws, REF–152
  - fputwc, REF–167
  - fputws, REF–168
  - fwide, REF–191
  - fwprintf, REF–192
  - fwscanf, REF–195
  - getwc, REF–228
  - getwchar, REF–229
  - iswalnum, REF–263
  - iswalpna, REF–264
  - iswcntrl, REF–265
  - iswctype, REF–266
  - iswdigit, REF–268
  - iswgraph, REF–269
  - iswlower, REF–270
  - iswprint, REF–271
  - iswpunct, REF–272
  - iswspace, REF–273

#### Wide-character functions (cont'd)

- iswupper, REF–274
- iswxdigit, REF–275
- mbrlen, REF–306
- mbrtowc, REF–307
- mbsinit, REF–311
- mbsrtowcs, REF–312
- putwc, REF–383
- putwchar, REF–384
- swprintf, REF–528
- swscanf, REF–529
- towctrans, REF–552
- towlower, REF–553
- towupper, REF–554
- ungetwc, REF–566
- vfwprintf, REF–589
- vfwscanf, REF–591
- vswprintf, REF–596
- vswscanf, REF–598
- vwprintf, REF–599
- vscanf, REF–600
- wcrtomb, REF–612
- wcscat, REF–613
- wcschr, REF–615
- wcscmp, REF–617
- wcscoll, REF–618
- wscspy, REF–619
- wscscspn, REF–620
- wcsftime, REF–622
- wcslen, REF–628
- wcsncat, REF–629
- wcsncmp, REF–631
- wcsncpy, REF–632
- wcspbrk, REF–633
- wcsrchr, REF–634
- wcsrtombs, REF–636
- wcsspn, REF–638
- wcsstr, REF–640
- wcstod, REF–641
- wcstok, REF–643
- wcstol, REF–646
- wcstoul, REF–649
- wcswcs, REF–652
- wcswidth, REF–654
- wcsxfrm, REF–655
- wctob, REF–658
- wctrans, REF–660
- wctype, REF–661
- wcwidth, REF–664
- wmemchr, REF–665
- wmemcmp, REF–666
- wmemcpy, REF–667
- wmemmove, REF–668
- wmemset, REF–669
- wprintf, REF–670
- wscanf, REF–676

Wide character I/O  
    program example, 2–23  
Wide-character I/O, 2–6  
winch function, REF–241  
winsch function, REF–246  
winsertrl function, REF–247  
winsstr function, 6–2, REF–248  
wmemchr function, REF–665  
wmemcmp function, REF–666  
wmemcpy function, REF–667  
wmemmove function, REF–668  
wmemset function, REF–669  
wmove function, REF–332  
wprintf function, REF–670  
wprintw function, REF–377  
wrapok function, REF–672  
wrefresh function, REF–398  
write function, REF–673

write socket routine, A–85  
writev function, REF–674  
wscanf function, REF–676  
wscanw function, REF–409  
wsetattr function, 6–2, REF–414  
wstandend function, REF–468  
wstandout function, REF–469

## X

---

\_XOPEN\_SOURCE macro, 1–19  
\_XOPEN\_SOURCE\_EXTENDED macro, 1–19

## Y

---

y0 function, REF–677  
y1 function, REF–677  
yn function, REF–677