



MetaWare® C

Library Reference

2133-024

MetaWare® C Library Reference

© 1990-2007 ARC® International. All rights reserved.

ARC International

North America	Europe
3590 N. First Street, Suite 200	Verulam Point
San Jose, CA 95134 USA	Station Way
Tel. +1-408-437-3400	St Albans, AL1 5HE
Fax +1-408-437-3401	UK
	Tel.+44 (0) 20-8236-2800
	Fax +44 (0) 20-8236-2801

www.ARC.com

ARC Confidential and Proprietary Information

Notice

This document, material and/or software contains confidential and proprietary information of ARC International and is protected by copyright, trade secret, and other state, federal, and international laws, and may be embodied in patents issued or pending. Its receipt or possession does not convey any rights to use, reproduce, disclose its contents, or to manufacture, or sell anything it may describe. Reverse engineering is prohibited, and reproduction, disclosure, or use without specific written authorization of ARC International is strictly forbidden. ARC and the ARC logotype are trademarks of ARC International.

The product described in this manual is licensed, not sold, and may be used only in accordance with the terms of a License Agreement applicable to it. Use without a License Agreement, in violation of the License Agreement, or without paying the license fee is unlawful.

Every effort is made to make this manual as accurate as possible. However, ARC International shall have no liability or responsibility to any person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by this manual, including but not limited to any interruption of service, loss of business or anticipated profits, and all direct, indirect, and consequential damages resulting from the use of this manual. ARC International's entire warranty and liability in respect of use of the product are set forth in the License Agreement.

ARC International reserves the right to change the specifications and characteristics of the product described in this manual, from time to time, without notice to users. For current information on changes to the product, users should read the "readme" and/or "release notes" that are contained in the distribution media. Use of the product is subject to the warranty provisions contained in the License Agreement.

Licensee acknowledges that ARC International is the owner of all Intellectual Property rights in such documents and will ensure that an appropriate notice to that effect appears on all documents used by Licensee incorporating all or portions of this Documentation.

The manual may only be disclosed by Licensee as set forth below.

- Manuals marked "ARC Confidential & Proprietary" may be provided to Licensee's subcontractors under NDA. The manual may not be provided to any other third parties, including manufacturers. Examples—source code software, programmer guide, documentation.
- Manuals marked "ARC Confidential" may be provided to subcontractors or manufacturers for use in Licensed Products. Examples—product presentations, masks, non-RTL or non-source format.
- Manuals marked "Publicly Available" may be incorporated into Licensee's documentation with appropriate ARC permission. Examples—presentations and documentation that do not embody confidential or proprietary information.

U.S. Government Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or in similar or successor clauses in the FAR, or the DOD or NASA FAR Supplement.

CONTRACTOR/MANUFACTURER IS ARC International I. P., Inc., 3590 N. First Street, Suite 200, San Jose, California 95134.

Trademark Acknowledgments

ARC-Based, ARC-OS Changer, ARCanGEL, ARCform, ARChitect, ARCompact, ARtangent, BlueForm, CASSEIA, High C/C++, High C++, iCon186, IPShield, MetaDeveloper, the MQX Embedded logo, Precise Solution, Precise/BlazeNet, Precise/EDS, Precise/MFS, Precise/MQX, Precise/MQX Test Suites, Precise/MQXsim, Precise/RTCS, Precise/RTCSsim, RTCS, SeeCode, TotalCore, Turbo186, Turbo86, V8 µ-RISC, V8 microRISC, and VAutomation are trademarks of ARC International. ARC, the ARC logo, High C, MetaWare, MQX and MQX Embedded are registered under ARC International. All other trademarks are the property of their respective owners.

2133-024 August 2007

Contents

Chapter 1 — Before You Begin	15
Technical Support	16
About This Book	16
Where to Go for More Information	16
The MetaWare C Run-Time Library at a Glance	17
Document Conventions	17
Notes	17
Cautions	17
Names and Terms	17
Regular Expressions	18
Chapter 2 — How to Use the Library	21
Linking with Library Files	22
Including Header Files	22
Declaring a Library Function	22
Return Type of an Undeclared Function	22
Required Header Files	22
Functions and Arguments	23
Arguments to Functions Declared with Prototypes	23
Arguments to Functions Called without Being Declared	23
Characters as Arguments	23
Characters as Return Values	23
Using Macros	24
Functions versus Macros that Implement a Function	24
Function Availability for UNIX and Embedded Targets	24
Reentrancy	25
How This Manual Documents Reentrancy	25
Checking errno in a Reentrant Environment	25
Reentrancy and the 386 Family Floating-Point Unit	26
Chapter 3 — The Header Files	27
Header File Listings	28
alloca.h	28
assert.h	28
bios.h	28
conio.h	28
ctype.h	29
direct.h	30
dirent.h	30
dos.h	30
errno.h	31
fcntl.h	31
float.h	31
io.h	32

limits.h	33
locale.h	33
locking.h	33
malloc.h	34
math.h	34
memory.h	36
_na.h	36
process.h	36
search.h	37
setjmp.h	37
share.h	37
signal.h	38
sizet.h	38
stat.h	38
stdarg.h	39
stddef.h	39
stdio.h	40
stdlib.h	42
string.h	44
time.h	46
timeb.h	47
types.h	47
utime.h	47
varargs.h	48

Chapter 4 — Function Reference 49

Function Listing Format	50
Function Listings	51
abort()	51
abs()	51
_abs()	52
access() and _access()	53
acos()	54
acosf()	55
acosh() and _acosh()	56
alloca() and _alloca()	57
asctime() and _asctime()	58
asin()	59
asinf()	60
asinh() and _asinh()	61
assert()	61
atan()	62
atan2()	63
atan2f()	65
atanf()	66
atanh() and _atanh()	67
atexit()	68
atof()	69
atoi()	69
atol()	70

_atold()	71
_bprintf()	72
bsearch()	73
cabs() and _cabs()	74
calloc()	75
ceil()	75
ceilf()	76
cgets() and _cgets()	77
chdir() and _chdir()	78
chmod() and _chmod()	79
chszie() and _chszie()	80
clearerr()	81
clock()	81
close() and _close()	82
_closedir()	83
cos()	84
cosf()	85
cosh()	86
coshf()	87
cprintf() and _cprintf()	88
cputs() and _cputs()	88
creat() and _creat()	89
_crotl() and _crotr()	91
cscanf() and _cscanf()	91
ctime() and _ctime()	92
dieetomsbin() and _dieetomsbin()	93
difftime()	95
_disable()	95
div()	96
_div0()	96
dmsbintoieee() and _dmsbintoieee()	97
dup() and _dup()	98
dup2() and _dup2()	98
ecvt() and _ecvt()	99
_enable()	100
eof() and _eof()	101
exit()	101
_exit()	102
exp()	103
expf()	104
fabs()	104
fabsf()	105
fclose()	106
fcloseall() and _fcloseall()	107
fcvt() and _fcvt()	107
fdopen() and _fdopen()	108
feof()	109
ferror()	110
fflush()	111

fgetc()	112
fgetchar() and _fgetchar()	112
fgetpos()	113
fgets()	114
fieetomsbin() and _fieetomsbin()	115
filelength() and _filelength()	116
fileno() and _fileno()	117
_findclose() and _findfirst() and _findnext()	118
floor()	119
floorf()	119
flushall() and _flushall()	120
fmod()	121
fmodf()	121
fmsbintoieee() and _fmsbintoieee()	122
fopen()	123
FP_OFF(), FP_SEG() and _FP_OFF(), _FP_SEG()	125
fprintf()	125
fputc()	126
fputchar() and _fputchar()	127
fputs()	128
fread()	129
free()	130
freopen()	130
frexp()	131
frexpf()	132
fscanf()	132
fseek()	133
fsetpos()	134
ftell()	135
ftime() and _ftime()	135
_fullpath()	136
fwrite()	137
gcvt() and _gcvt()	138
getc()	139
getchar()	139
getcwd() and _getcwd()	140
getenv()	141
getpid() and _getpid()	142
gets()	143
getw() and _getw()	143
gmtime() and _gmtime()	144
_heapchk()	145
_heapset()	146
_heapwalk()	146
hypot() and _hypot()	147
_initcopy()	148
_inline()	148
_invalidate_dcache()	149
_invalidate_icache()	150
isalnum()	150

isalpha()	151
isascii() and _isascii()	151
isatty() and _isatty()	152
iscntrl()	153
isdigit()	154
isgraph()	154
islower()	155
_isodigit()	155
isprint()	156
ispunct()	157
isspace()	157
isupper()	158
isxdigit()	159
itoa() and _itoa()	159
j0(), j1(), jn() and _j0(), _j1(), _jn()	160
labs()	161
_ldecvt()	162
ldexp()	163
ldexpf()	163
_ldfcvt()	164
ldiv()	165
lfind() and _lfind()	165
localeconv()	167
localtime() and _localtime()	167
log()	168
log10()	169
log10f()	170
log2() and _log2()	170
logf()	171
longjmp()	172
_lrotl() and _lrotr()	173
lsearch() and _lsearch()	174
lseek() and _lseek()	175
ltoa() and _ltoa()	176
_makepath()	177
malloc()	178
matherr(), matherrx() and _matherr(), _matherrx()	179
_max()	180
mblen()	181
mbstowcs()	182
mbtowc()	183
memccpy() and _memccpy()	183
memchr()	184
memcmp()	185
memcpy()	186
memicmp() and _memicmp()	187
memmove()	188
memset()	189
_min()	190

mkdir() and _mkdir()	190
mktemp() and _mktemp()	191
mktimes()	192
modf()	193
modff()	194
_msize()	194
offsetof()	195
open() and _open()	196
_opendir()	198
perror()	199
pow()	199
powf()	200
printf()	201
putc()	207
putch() and _putch()	208
putchar()	208
putenv() and _putenv()	209
puts()	210
putw() and _putw()	211
qsort()	212
raise()	213
rand()	214
read() and _read()	214
_readdir()	215
realloc()	216
remove()	217
rename()	218
rewind()	219
_rewinddir()	219
rmdir() and _rmdir()	220
_rmemcpy()	221
_rotl() and _rotr()	222
_rstrcpy()	223
_rstrncpy()	224
scanf()	225
_searchenv()	231
_searchstr()	231
setbuf()	232
setjmp()	233
setlocale()	234
_set_matherr()	235
_setmode()	236
setvbuf()	237
_setvect1()	238
_setvect2()	239
signal()	239
sin()	241
sinf()	242
sinh()	243
sinhf()	244

_sleep()	245
spawn*() and _spawn*()	246
_splitpath()	248
sprintf()	249
sqrt()	250
sqrtf()	250
rand()	251
_srotl() and _srotr()	252
sscanf()	252
stat() and _stat()	253
strcat()	254
_strcats()	255
strchr()	255
strcmp()	256
strcmpi() and _strcmpi()	257
strcoll()	258
strcpy()	259
strcspn()	260
_strdate()	260
strdup() and _strdup()	261
strerror() and _strerror()	262
strftime()	263
strcmp() and _strcmp()	264
strlen()	265
strlwr() and _strlwr()	266
strncat() and _strncat()	266
strncmp()	267
strncpy()	268
strnicmp() and _strnicmp()	269
strnset() and _strnset()	270
strupr()	271
strrchr()	271
strrev() and _strrev()	272
strset() and _strset()	273
strspn()	274
strstr()	275
_strtime()	275
strtod()	276
strtok()	277
strtol()	278
strtoul()	279
strupr() and _strupr()	280
strxfrm()	281
swab() and _swab()	281
system()	282
tan()	283
tanf()	284
tanh()	285
tanhf()	286

tell() and _tell()	287
_tempnam()	288
time()	289
tmpfile() and _tmpfile()	289
tmpnam()	290
tolower()	291
_tolower()	292
toupper()	292
_toupper()	293
tzset() and _tzset()	294
_udiv0()	295
ultoa() and _ultoa()	296
umask() and _umask()	296
ungetc()	297
ungetch() and _ungetch()	298
unlink() and _unlink()	299
utime() and _utime()	300
utoa() and _utoa()	301
va_arg()	301
va_end()	302
va_start()	303
_vbprintf()	304
_vdmemcpy()	305
vfprintf()	305
vfscanf() and _vfscanf()	306
_vmemset()	308
vprintf()	308
_vsdmemcpy()	309
_vsmemcpy()	310
vscanf() and _vscanf()	311
vsprintf()	312
vsscanf() and _vsscanf()	313
wcstombs()	314
wctomb()	314
write() and _write()	315
y0(), y1(), yn() and _y0(), _y1(), _yn()	316

Chapter 5 — Constants and Globals

319

Listing of Constants and Global Variables	320
A*	320
BUFSIZ	321
CHAR_*	321
CLOCKS_PER_SEC	322
COM*	322
--DATE--	323
_daylight	324
DBL_*	324
_E	326
EDOM	326
EOF	326

ERANGE	327
errno	327
EXIT_FAILURE and EXIT_SUCCESS	329
__FILE__	329
FILENAME_MAX	330
FLT_*	330
FOPEN_MAX	331
_HEAP*	332
--HIGHC--	333
HUGE_VAL	333
INT_*	334
_INTERRPT	334
_IOFBF and _IOLBF and _IONBF	334
KEYBRD*	335
LC_*	336
LDBL_*	337
--LINE--	338
LK_* and _LK_*	339
LONG_*	339
L_tmpnam	340
MAX*	340
_MAXSTRING	341
MB_CUR_MAX	342
MB_LEN_MAX	342
NULL	343
O_* and _O_*	343
P*	345
_PI	346
PRINTER*	346
RAND_MAX	347
SCHAR_*	347
SEEK_CUR and SEEK_END and SEEK_SET	348
SH*	348
SHRT_*	349
_S_I*	350
SIG_DFL and SIG_ERR and SIG_IGN	350
SIG*	351
stderr and stdin and stdout	353
SW_*	353
_threadid	355
--TIME--	355
timezone	356
TMP_MAX	356
U*_MAX	357
Chapter 6 — Typedefs and Structs	359
Listing of Typedefs, Structs, and Unions	360
BYTEREGS	360
clock_t	360

complex	361
_dirent	361
diskfree_t	362
div_t	363
dosdate_t	363
DOSERROR	364
dostime_t	364
DWORDREGS	365
exception	366
FILE	367
finddata_t	368
find_t	369
fpos_t	370
_HEAPINFO	370
jmp_buf	371
Iconv	371
ldiv_t	373
ptrdiff_t	374
REGS	374
sig_atomic_t	375
size_t	375
SREGS	376
stat	376
timeb	377
time_t	378
tm	379
utimbuf	380
va_list	381
wchar_t	381
WORDREGS	382

Appendix A — Input and Output 385

Logical Data Streams	386
Buffered Streams	386
File Pointers	386
Opening and Closing Files	386
Text and Binary Streams	386
Errors	387
Variable errno	387
Error Flag	387
Flushing Buffers	387
End of File	388
End-of-File Flag	388
End of File on DOS and Windows NT Targets	388

Appendix B — Unicode Functions 389

MetaWare C Unicode Functions	390
What is Unicode?	390
Are MetaWare C Unicode Functions Available on my Target System?	390
Using Unicode Functions	391

printf and scanf Format Conversions	391
Printing ASCII Characters with Unicode Functions	391
Printing Unicode Characters with ASCII Functions	391
Dual-Mode Macros	392
Unicode Function in assert.h	392
Unicode Functions in ctype.h	392
Unicode Functions in stdio.h	393
Unicode Functions in stdlib.h	394
Unicode Functions in string.h	395
Unicode Functions in time.h	396
Appendix C — ANSI Compliance	397
ANSI-Compliant Functions	398
ANSI C99-Compliant Functions	402
Extra-ANSI Functions	403
Non-ANSI Functions	407
Appendix D — Reentrancy Status	411
Reentrant Functions	412
Not-Reentrant Functions	418
Appendix E — Function Families	423
ARC-Only Functions	424
The *printf Family of Functions	424
The *scanf Family of Functions	424
Transcendental Math Functions	424
Conversion Functions	425
Case Conversions	425
String-to-Numeric Conversions	426
Numeric-to-String Conversions	426
IEEE/Microsoft Binary Conversions	426
Multibyte/Wide-Character Conversions	426
int and long int Conversions	426
unsigned int and unsigned long Conversions	427
double and long double Conversions	427
float Conversions	427
I/O Functions	427
Standard I/O	427
Keyboard/console I/O	428
Character I/O	428
String I/O	428
File I/O	428
File-Related Functions	429
File-Access Functions	430
Get Information about a File, Directory, Handle, or Stream	430
Modify an Existing File or Directory	430
Find an Existing File	430
Pathname Functions	430
Create a File, a File Name, or a Directory	430
Duplicate a File Handle	431
Open a File	431

Read from a File	431
Write or Print to a File	431
File-Position Functions (and Macros)	431
File-Buffering Functions	432
Close a File or Directory Stream	432
Delete a File or Directory	432
String-Handling Functions	432
Comparing Strings	432
Concatenating or Appending Strings	433
Converting, Transforming, or Formatting Strings	433
Converting Other Data to Strings	433
Copying Strings	433
Determining String Length	433
Filling Strings	433
Finding Characters in Strings	434
Memory Allocation and Deallocation Functions	434
Time-Related Functions	434

Chapter 1 — Before You Begin

In This Chapter

- [Technical Support](#)
- [About This Book](#)
- [The MetaWare C Run-Time Library at a Glance](#)
- [Document Conventions](#)

Technical Support

We encourage customers to visit our Technical Support site for technical information before submitting support requests via telephone. The Technical Support site is richly populated with the following:

ARCSolve — Frequently asked questions

ARCSpeak — Glossary of terms

White papers — Technical documents

Datasheets — ARC product flyers

Known Issues — Known bugs and workarounds.

Training — Future ARC Training sessions

Support — Entry point to sending in a query

Downloads — Access to useful scripts and tools

You can access this site via our corporate website at <http://www.ARC.com>, or you can access the site directly: <http://support.ARC.com>.

Note that you must register before you can submit a support request via our Extranet site.

About This Book

This *MetaWare ® C Library Reference* describes the MetaWare C Run-Time library. We assume you are familiar with the C language, and with the operating system and machine on which the MetaWare C Compiler and Run-Time Library will be used.

This book contains the following topics:

- [How to Use the Library](#)
- [The Header Files](#)
- [Function Reference](#)
- [Constants and Globals](#)
- [Typedefs and Structs](#)
- [Input and Output](#)
- [Unicode Functions](#)
- [ANSI Compliance](#)
- [Reentrancy Status](#)
- [Function Families](#)

Where to Go for More Information

The release note or `readme` file describes any special files and provides information that was not available at the time this *MetaWare C Library Reference* was published.

The *I/O Streams Library Reference* documents the C++ input and output streams.

The *MetaWare C/C++ Language Reference* describes the syntax and semantics of the C and C++ language and the MetaWare C/C++ extensions to the C and C++ language.

The *MetaWare C/C++ Programmer's Guide* contains all the system-specific information you need to use the compiler.

For a detailed list of other relevant documents, see the "Where to Go for More Information" section in the *MetaWare C/C++ Programmer's Guide*.

The MetaWare C Run-Time Library at a Glance

The MetaWare C Run-Time Library is a collection of function, macro, and type declarations organized into header files according to ANSI Standard C Library areas, such as mathematics and string handling.

The MetaWare C Run-Time Library is an ANSI Standard C library with extensions. The macros, function names and parameters, and types are those dictated by the ANSI C Standard. The library conforms to the ANSI C Standard (ANSI document X3J11/90-014); the single-precision math functions conform to the ISO/ANSI C99 Standard. The library also provides additional functions not described in the ANSI C Standard.

Document Conventions

Notes

Notes point out important information

NOTE Non-strict semaphores do not have priority inheritance.

Cautions

Cautions tell you about commands or procedures that could have unexpected or undesirable side effects or could be dangerous to your files or your hardware.

CAUTION If you modify MQX data types, some tools may not operate properly.

Names and Terms

The following terms appear throughout this manual.

ANSI compliant

Refers to a language feature that conforms to standards published by the American National Standards Institute, which is responsible for the ANSI Standard C language and library definition.

C99 compliant

Refers to a function that conforms to the ISO/ANSI C99 Standard.

DOS

Target platforms running 32-bit Extended DOS or Windows 3.x. Compare to the term [Windows NT](#).

errno

An **int** variable declared in `errno.h` and set to 0 (zero) at program start-up. For more information, see global variable [errno](#) in [Constants and Globals](#) on page 319 and [Variable errno](#) on page 387.

Extra-ANSI

Refers to a language feature that does not conform to, yet does not violate, the ANSI Standard C language and library definition.

Functions, types, and macros in the library that are not required by the ANSI C Standard, yet do not violate ANSI naming conventions, have names that begin with an underscore (_), and are designated Extra-ANSI.

install_dir

Short-hand notation for the parent directory in which you have installed MetaWare C. Used in the function reference sections to direct you to the installed examples directory.

Non-ANSI

Refers to a language feature that does not conform to the ANSI Standard C language and library definition.

For compatibility with previous products, “Non-ANSI” versions of some of these functions can be called *without* the underscore prefix. Such functions execute identically with either name. See option **-Hnolib** in the *MetaWare C/C++ Programmer’s Guide* for more information.

NUL

A name for the character ‘\0’, the string termination character, whose value is 0 (zero). Note that **NUL** is different from **NULL**, a macro that expands to a representation of the null data pointer.

NULL

A macro that expands to a value that is assignment-compatible with any data-pointer type and compares equal to the constant 0 (zero). **NULL** is therefore suitable as a representation of the null pointer.

Note that **NULL** is not appropriate as the terminating character of a string, because the size of a pointer is not the same as the size of the character **NUL**. In addition, on some systems, function pointers and data pointers are not the same size. **NULL** is defined in `stddef.h`, `stdio.h`, and `stdlib.h`.

Null pointer

Any pointer that compares equal to 0 (zero).

size_t

A type specifier; type of the result of `sizeof`. **size_t** is defined in `stddef.h`, `stdio.h`, `stdlib.h`, and `string.h`. See [size_t](#) in [Chapter 6 — Typedefs and Structs](#).

String

Refers to a pointer to the first character of a sequence of characters terminated with **NUL**.

Windows NT

Target platforms running Windows NT or Windows 95 on 32-bit Intel 386 family processors. Compare to the term [DOS](#).

Regular Expressions

The library contains functions that handle sequences of characters. Several of these functions expect the sequence to be of some particular form. A simple example of such a form is a digit sequence. There is an infinite number of such sequences, but the regular expression `Digit*` describes them all succinctly.

Regular expressions, which are used throughout this manual, provide a way to specify the order and number of characters in a sequence.

Terms Used in Regular Expressions

The following terms are used in regular-expression notation throughout this manual:

Whitespace (or Wspace)

A set of characters including tabs, spaces, and newline characters. Whitespace is that set of characters tested for by the function `isspace()` declared in header file `ctype.h`.

Sign

A plus or minus sign ('+' or '-'). These characters must be quoted.

Digit (or Dgt)

Any decimal digit; that is, characters in the range 0 through 9. Individual `Digit` digits must be quoted.

Odigit

Any octal digit, that is, any `Digit` except 8 or 9. Individual `Odigit` digits must be quoted.

Hexdigit

Any hexadecimal digit which is a `Digit`, or any of the characters 'a' through 'f' (or 'A' through 'F' — case is not significant in numbers) having decimal values in the range 10 through 15, respectively. Individual `Hexdigit` digits must be quoted.

Modifiers Used in Regular Expressions

The descriptive power of these expressions comes from modifiers that act on the elements (characters and names) of an expression. The modifiers are as follows:

Symbol	Meaning
*	Zero or more
+	One or more
?	Zero or one
	Alternation
-	Set subtraction

The modifiers are not quoted, and parentheses are used for grouping. For example, this is a regular expression describing a sequence composed of zero or more 'k' characters:

'k'*

and this is a regular expression describing a sequence composed of zero or more asterisk characters:

'*'*

Defining a Name to Represent a Regular Expression

A name *Nm* followed by a right arrow (->), followed by a regular expression *Exp*, and terminated with a semicolon (;), defines *Nm* to represent *Exp*, as illustrated here:

Nm -> *Exp*;

Thus `Digits`, `Odigits`, and `Hexdigits` can be precisely defined as follows:

```
Digit      -> '0' | '1' | '2' | '3' | '4' |
                  '5' | '6' | '7' | '8' | '9';
Digits     -> Digit+;
Odigit    -> Digit - '8' - '9' ;
Odigits   -> Odigit+;
Hexletter  -> 'a' | 'b' | 'c' | 'd' | 'e' | 'f' |
                  'A' | 'B' | 'C' | 'D' | 'E' | 'F';
Hexdigit  -> Digit | Hexletter;
Hexdigits -> Hexdigit+;
```

A digit sequence with an optional leading sign could be named **Integer** and defined as follows:

```
Integer    -> ('+' | '-')?Digit+;
```

The format sequence that is the first argument to **printf()** and the second argument to **fprintf()** can be described as follows:

```
Format     -> (any - '%' | Conversion_spec)*;
```

Here, any refers to any character. The conversion specification (**Conversion_spec**) is illustrated here by an extract from the description of function **printf()**:

```
Conversion_spec -> '%' Flag* Field_width ? Precision ?
                           Size ? C_char;
Flag        -> '-' | '+' | ' ' | '#' | '0' ;
Field_width -> ((Digit - '0') Digits) | '*' ;
Precision   -> '.' (Digits | '*');
Size        -> 'h' | 'l' | 'L' ;
C_char      -> 'd' | 'i' | 'o' | 'u' | 'x' | 'X' |
                  'e' | 'E' | 'f' | 'g' | 'G' |
                  'c' | 's' | 'p' | 'n' | '%';
```

The following are valid calls to **printf()**. The string that is the first argument to **printf()** is a valid **Format** string.

```
printf("Hello world.");
printf("Hello %s.", "world");
printf("%d * %hd = %ld", 2, 3, 6L);
printf("%05d * %.5f = %+.#07.*LG", 2, 3.14, 8, 6.28L);
printf("I am behind you %d%%.", 1000);
```

The following are not valid calls to **printf()**. The string that is the first argument to **printf()** is not a valid **Format** string.

```
printf("Hello world.%");
/* Missing C_char */
printf("Hello %k.", "world");
/* k is not a C_char */
printf("%d * %h = %hld", 2, 3, 6);
/*
 * h is not a C_char.
 * Also, only one of 'h' or 'l' can be
 * generated from Size.
*/
printf("%105d * %f.5 = %7+.#0.*LG", 2, 3.14, 6.28);
/*
 * Size (l) must follow Field_width (05)
 * C_char (f) must follow Precision (.5)
 * Field_width (7) must follow Flags(+#0)
 */
printf("I am behind you %d%%.", 1000);
/* Missing C_chars */
```

Chapter 2 — How to Use the Library

In This Chapter

- [Linking with Library Files](#)
- [Including Header Files](#)
- [Functions and Arguments](#)
- [Using Macros](#)
- [Function Availability for UNIX and Embedded Targets](#)
- [Reentrancy](#)

Linking with Library Files

Once the compiler has finished compiling your source code, the driver invokes the linker to link the appropriate run-time libraries with the object modules generated by the compiler. These libraries contain routines to set up the environment, manage the heap, perform I/O, and handle other standard operations.

For detailed information about linking run-time libraries, see the *MetaWare C/C++ Programmer's Guide* and the *ELF Linker and Utilities User's Guide*.

Including Header Files

Header files contain declarations for the functions, constants, and macros provided by the MetaWare C Library, and additional information not contained in the documentation. Not all files listed here are provided for all platforms; some are specific to a particular operating environment. See [The Header Files](#) on page 27 for information about individual header files.

NOTE Header files can be included in any order, and they can be included more than once with no ill effects.

Declaring a Library Function

If your C program uses a library function, you declare the function by including the appropriate header file in your source file. However, the ANSI Standard C language permits references to functions that are not declared within the compilation unit containing the reference.

This is possible because a function can be compiled separately from a compilation unit in which it is called, and a resolution between the function call and the function can be made at link time.

Return Type of an Undeclared Function

The return type of an undeclared function is assumed to be **int**. The library has been carefully designed so that any library function satisfying both of the following constraints can be called without being declared:

- The function takes arguments of *scalar* type.
- The function returns either nothing or a result of an integral type.

A danger inherent in this capability is that functions that do not satisfy these constraints can also be called without being declared, but because the compiler assumes the constraints to hold, unexpected behavior might result at run time. To correctly use a function that does not satisfy the stated constraints, either include the header file that contains its declaration (safe) or duplicate the declaration (unreliable).

Required Header Files

In the Synopsis section of each function description, this line appears if the function cannot be called safely without either including the header or duplicating some of the text from the header:

```
#include <header_file_name> /* Required */
```

This limitation might be because the function does not satisfy the constraints mentioned previously, or because the function takes or returns an argument of a type declared in a header file.

Despite C's function-calling flexibility, it is a good idea to always include the header file. Such inclusion provides compile-time protection against passing the wrong number or type of arguments to these functions.

Unlike functions, references to types and macros must be resolved during compilation. If a type or macro provided by this library is used in a program, either the header file that defines it must be included, or the definition must be duplicated in the source code.

Functions and Arguments

The ANSI Standard C language provides a syntax for function declarations that allows the type and number of arguments to be checked against the type and number of declared parameters. Such a declaration is called a *prototype*.

Prototypes are used to declare library functions in the header files. The library is designed as much as possible to allow library functions to be called without being declared, which has an impact on the declarations of some functions.

Arguments to Functions Declared with Prototypes

Arguments to functions that are declared with prototypes must be compatible with the types of the declared parameters, in the same sense that the right part of an assignment expression must be compatible with its left part.

The arguments are passed with the same conversions that occur with assignment — that is, no conversion occurs if the types are the same. If the types are not the same, some arithmetic conversion might occur, such as from **int** types to **float** types.

Arguments to Functions Called without Being Declared

If a function is called without being declared, or is declared without a prototype, the following conversion of the function's argument(s) occurs:

- If the argument is a **signed char** or **signed short**, it is converted to **signed int**.
- If the argument is an **unsigned char** or **unsigned short**, it is converted to **unsigned int**.
- If the argument is a **float**, it is converted to **double**.
- Otherwise, no conversion occurs.

Characters as Arguments

Some library functions take characters as arguments, but the corresponding parameters are declared as **int** rather than **char**. This is because **char** types are widened to **int** types when passed to non-prototyped functions.

If such a parameter *c* were declared to be type **char** and the function were called without being declared, *c* would be passed as an **int** and the behavior would be undefined. For consistency, all such functions take **int** types rather than **char** types, even if they are likely to be used only if the appropriate header file is included.

Characters as Return Values

Likewise, functions that return character values are declared to return **int** types so they can be used without being declared, because undeclared functions are assumed to return **int**. In addition, a number of such functions must be able to return **EOF**, a macro declared in **stdio.h** that expands to an **int** literal.

Using Macros

Library functions implemented as macros (such as `f.eof()`) are documented using the syntactic form of functions. The documentation uses the parameter and return types that would be appropriate if the macro were a true function, even though macros are not typed.

The library contains two kinds of macros: those that expand to a constant or variable reference, and those that implement a provided function.

NOTE For a macro to be used, the header file containing its definition must be included in the source file being compiled.

Functions versus Macros that Implement a Function

Several functions are provided with two implementations: one as a macro and one as a function. This is mentioned in the description of each such function.

Usually, the effect of a function is identical to the effect of the corresponding macro. A function call evaluates each argument once, while a macro can evaluate its arguments more than once. Therefore, the effect of a function call might differ from the effect of the corresponding macro expansion if an argument has side effects. For each macro provided, if any arguments can be multiply evaluated, that fact is documented.

In a given instance, it might be desirable to choose a function over a macro, or vice versa. To access a provided macro, simply include the header file. If the header file is not included, a reference to the function name references the function.

It might be desirable to include the header file and still reference the function instead of the macro, because each header file typically provides a number of functions.

To reference the function while including the header file, include this line in your source code:

```
#undef function_name
```

This directive removes the definition of *function_name* as a macro, so that all subsequent instances of *function_name* refer to the function. To reference the macro, you must duplicate its definition.

You must also include such an **undef** directive in the source file any time a name that is defined as a macro must be redefined as a macro or declared as the name of an object in the program.

If a macro has parameters, substitution on that macro occurs only if it appears immediately followed by the parenthesized parameter list. Thus, surrounding the name with parentheses disables the substitution and forces a call to the function. For example, `getc(F)` invokes a macro but `(getc)(F)` calls a function. This allows those functions with arguments to be called while the corresponding macros retain their definitions.

Function Availability for UNIX and Embedded Targets

All ANSI-required functions and macros are discussed in [Function Reference](#) on page 49. However, we do not necessarily provide all these objects for all architectures. When an object is not provided for a specific architecture, you must rely on the target environment to provide that object.

In the System issues section of the object's description, you will see one of the following statements regarding the object's availability:

- On UNIX and embedded systems, this function can be provided by the system library or some third-party library.
- On embedded systems, this function can be provided by some third-party library.
- On UNIX, this function can be provided by the system library.

Reentrancy

A *reentrant routine* is one that can be used by two or more independent programs at the same time. A reentrant routine never modifies itself and never shares data with any other instance of itself. Examples of code that can simultaneously invoke more than one instance of the same routine are recursive functions, calls by another process, and interrupt service routines.

Subroutines shared between tasks of a real-time operating system can context switch on a timer tick during execution, and schedule another task that invokes the same subroutine. If two or more instances of the same subroutine are invoked and then interrupted during execution, each invocation must store its data in a different area of RAM. Routines that use global or static data violate this rule.

The ANSI Standard C library specification mandates the use of global and static data. This means that, by definition, the ANSI Standard C library as a whole is not reentrant. However, MetaWare C provides many individual ANSI compliant, Extra-ANSI, and Non-ANSI routines that do meet the requirements for reentrancy.

How This Manual Documents Reentrancy

Each function described in [Function Reference](#) on page 49 is labeled according to its reentrancy. Some functions listed as reentrant are only reentrant when used on certain platforms. Check the System issues section of each function description for details regarding reentrancy.

[Reentrancy Status](#) on page 411 lists the functions according to whether they are Reentrant or Not-Reentrant.

Note Any routine that uses a non-reentrant function becomes non-reentrant itself.

Checking `errno` in a Reentrant Environment

In a reentrant environment, checking `errno` after a math function is not safe. Many routines, such as floating-point routines, set the global variable `errno` when they encounter an error. This is not a violation of reentrancy unless you are using the value of `errno` to conditionalize execution of a section of code.

The following example illustrates how you would violate reentrancy using `errno`:

```
errno = 0;
double d = sin(x);
if (errno == 0) /* Unreliable check */
    /* ...Continue processing */
```

In this example, if function `sin()` set `errno` to a non-zero value, execution might have been interrupted by another program before the `errno` check. The program operating during the interrupt might also

call **sin()** and set **errno** back to 0 (zero). When the first routine returns after the interrupt, the **errno** check produces unreliable results.

Reentrancy and the 386 Family Floating-Point Unit

When you use native floating-point routines and floating-point library routines in an interrupt handler on an Intel 386 family platform, it is your responsibility to save and restore the state of the floating-point unit.

Math routines categorized as reentrant assume that you have performed the necessary saving, reinitializing, and restoring of the state of the floating-point unit with the **FSAVE**, **FINIT**, and **FRESTOR** floating-point instructions.

Chapter 3 — The Header Files

In This Chapter

- [Header File Listings](#)

Header File Listings

This section presents an alphabetical list of the header files.

alloca.h

Header file `alloca.h` defines the function [alloca\(\) and _alloca\(\)](#), which is used to allocate storage from the caller's stack frame.

`_alloca()` is actually implemented as a compiler intrinsic. However, this header file *must* be included before referencing the function `_alloca()`.

`_alloca()` is reentrant.

assert.h

Header file `assert.h` defines one macro, [assert\(\)](#), which puts diagnostics into a program. If a user-defined macro `NDEBUG` is defined at the point of inclusion of the header file `assert.h`, the [assert\(\)](#) macro has no effect. The [assert\(\)](#) macro is reentrant.

NOTE `assert.h` is an ANSI-defined header file.

For information about unicode equivalents, see [Unicode Function in assert.h](#) on page 392.

bios.h

Header file `bios.h` defines the functions, structures, and constants needed to access the BIOS service.

NOTE `bios.h` is available only for DOS targets.

Structures and Unions

`struct BYTEREGS` — [Contents of 386 family hardware byte registers](#)

`struct SREGS` — [Contents of 386 family segment registers](#)

`struct WORDREGS` — [Contents of 386 family hardware word registers](#)

`union REGS` — [Contents of 386 family word- and byte-register structures](#)

Constants

`COM_*` — [Constants for BIOS asynchronous communication](#)

`KEYBRD_*` — [Keyboard services for function bios_keybrd\(\)](#)

`PRINTER_*` — [BIOS printer services](#)

conio.h

Header file `conio.h` defines functions for console input and output.

Note conio.h is available only for DOS and Windows NT targets.

Functions

- [cgets\(\) and _cgets\(\), cgets\(\) and _cgets\(\)](#) — [Read a line of text \(string\) from the keyboard](#)
- [cprintf\(\) and _cprintf\(\), cprintf\(\) and _cprintf\(\)](#) — [Print to screen](#)
- [cputs\(\) and _cputs\(\), cputs\(\) and _cputs\(\)](#) — [Write a string to the screen](#)
- [cscanf\(\) and _cscanf\(\), cscanf\(\) and _cscanf\(\)](#) — [Read from standard input](#)
- [putch\(\) and _putch\(\), putch\(\) and _putch\(\)](#) — [Write a character to the screen](#)
- [ungetch\(\) and _ungetch\(\), ungetch\(\) and _ungetch\(\)](#) — [Push a character back into the input buffer](#)

ctype.h

Header file ctype.h defines functions for handling characters.

Note ctype.h is an ANSI-defined header file.

Functions

- [isalnum\(\)](#) — [Test for alphanumeric character](#)
- [isalpha\(\)](#) — [Test for alphabetic character](#)
- [isascii\(\) and _isascii\(\), isascii\(\) and _isascii\(\)](#) — [Test for ASCII character](#)
- [isctrl\(\)](#) — [Test for control character](#)
- [isdigit\(\)](#) — [Test for numeric character](#)
- [isgraph\(\)](#) — [Test for visible character](#)
- [islower\(\)](#) — [Test for lowercase alphabetic character](#)
- [isodigit\(\)](#) — [Test for octal numeric character](#)
- [isprint\(\)](#) — [Test for printable character](#)
- [ispunct\(\)](#) — [Test for punctuation character](#)
- [isspace\(\)](#) — [Test for whitespace character](#)
- [isupper\(\)](#) — [Test for uppercase alphabetic character](#)
- [isxdigit\(\)](#) — [Test for hexadecimal numeric character](#)
- [tolower\(\)](#) — [Convert to lowercase](#)
- [tolower\(\)](#) — [Convert to lowercase without testing for uppercase](#)
- [toupper\(\)](#) — [Convert to uppercase](#)
- [toupper\(\)](#) — [Convert to uppercase without testing for lowercase](#)

For information about unicode equivalents to these functions, see [Unicode Functions in ctype.h](#) on page 392.

direct.h

Header file direct.h defines functions for directory service.

NOTE direct.h is available only for DOS and Windows NT targets.

Functions

[chdir\(\)](#) and [_chdir\(\)](#), [chdir\(\)](#) and [_chdir\(\)](#) — [Change the current working directory](#)

[getcwd\(\)](#) and [_getcwd\(\)](#), [getcwd\(\)](#) and [_getcwd\(\)](#) — [Get full pathname of the current working directory](#)

[mkdir\(\)](#) and [_mkdir\(\)](#), [mkdir\(\)](#) and [_mkdir\(\)](#) — [Create a new directory](#)

[rmdir\(\)](#) and [_rmdir\(\)](#), [rmdir\(\)](#) and [_rmdir\(\)](#) — [Remove a directory](#)

dirent.h

Header file dirent.h defines functions and structures for POSIX directory operations.

NOTE dirent.h is available only for DOS and Windows NT targets.

Functions

[closedir\(\)](#) — [Close a POSIX directory stream](#)

[opendir\(\)](#) — [Open a POSIX directory stream](#)

[readdir\(\)](#) — [Read a POSIX directory stream](#)

[rewinddir\(\)](#) — [Reset a POSIX directory stream](#)

Structures

struct [dirent](#) — [A POSIX file name for _readdir\(\)](#)

dos.h

Header file dos.h defines DOS-related functions and services.

NOTE dos.h is available only for DOS targets.

Functions

[disable\(\)](#) — [Disable interrupts](#)

[enable\(\)](#) — [Enable interrupts](#)

[FP_OFF\(\)](#), [FP_SEG\(\)](#) and [_FP_OFF\(\)](#), [_FP_SEG\(\)](#), [FP_OFFSET\(\)](#), [FP_SEGMENT\(\)](#) and [_FP_OFFSET\(\)](#), [_FP_SEGMENT\(\)](#) — [Get offset and segment of a pointer](#)

Constants

[A_*](#) — [File attributes](#)

[INTERRPT](#) — [Designation of a DOS interrupt handler](#)

Structures and Unions

struct BYTEREGS — [Contents of 386 family hardware byte registers](#)
struct diskfree_t — [Information about space on a disk drive](#)
struct dosdate_t — [Information about current DOS system date](#)
struct DOSERROR — [Detailed information about a DOS error](#)
struct dostime_t — [Information about current system time](#)
struct find_t — [Name and attributes of a file](#)
struct SREGS — [Contents of 386 family segment registers](#)
struct WORDREGS — [Contents of 386 family hardware word registers](#)
union REGS — [Contents of 386 family word- and byte-register structures](#)

errno.h

Header file `errno.h` defines the error-code values and the **int** variable `errno`. For a listing of the error-code values defined in `errno.h`, see [errno](#) in [Constants and Globals](#) on page 319.

Note `errno.h` is an ANSI-defined header file.

fcntl.h

Header file `fcntl.h` defines constants used by `open()` and `sopen()`. These constants control the operations that can be performed upon the resulting file handle.

Functions

[creat\(\) and _creat\(\), creat\(\) and _creat\(\)](#) — [Create an empty file](#)
[open\(\) and _open\(\), open\(\) and _open\(\)](#) — [Open a file handle with a pathname](#)

Constants

[O_* and _O_*](#) and [O_* and _O_*](#) — [File-open mode flags](#)

float.h

Header file `float.h` defines constants that constrain **float**, **double**, and **long double** representation.

Note `float.h` is an ANSI-defined header file.

Constants

[DBL_*](#) — [Minimum and maximum values for double calculations](#)
[FLOAT_*](#) — [Minimum and maximum values for float calculations](#)
[LDBL_*](#) — [Minimum and maximum values for long double calculations](#)

io.h

Header file `io.h` defines DOS file-service functions.

NOTE `io.h` is available only for DOS and Windows NT targets.

Functions

[access\(\) and accessl\(\), access\(\) and access\(\)](#) — [Test access rights of a file](#)
[chmod\(\) and chmodl\(\), chmod\(\) and chmod\(\)](#) — [Alter the permission setting for a file](#)
[chsize\(\) and chsizel\(\), chsize\(\) and chsize\(\)](#) — [Extend or truncate the length of a file](#)
[close\(\) and closel\(\), close\(\) and close\(\)](#) — [Close a file handle](#)
[creat\(\) and creatl\(\), creat\(\) and creat\(\)](#) — [Create an empty file](#)
[dup\(\) and dupl\(\), dup\(\) and dup\(\)](#) — [Duplicate a file handle](#)
[dup2\(\) and dup2l\(\), dup2\(\) and dup2\(\)](#) — [Duplicate one file handle into another](#)
[eof\(\) and eofl\(\), eof\(\) and eof\(\)](#) — [Test a file handle for end-of-file status](#)
[filelength\(\) and filelengthl\(\), filelength\(\) and filelength\(\)](#) — [Determine length of an open file](#)
[findclose\(\) and findfirst\(\) and findnext\(\), findclose\(\) and findfirst\(\) and findnext\(\), findclose\(\) and findfirst\(\) and findnext\(\)](#) — [Find a file whose name matches a specification](#)
[isatty\(\) and isattyl\(\), isatty\(\) and isatty\(\)](#) — [Identify character devices](#)
[lseek\(\) and lseekl\(\), lseek\(\) and lseek\(\)](#) — [Change the position in a file](#)
[mktemp\(\) and mktempl\(\), mktemp\(\) and mktemp\(\)](#) — [Create a unique file name \(does not create or open files\)](#)
[open\(\) and openl\(\), open\(\) and open\(\)](#) — [Open a file handle with a pathname](#)
[read\(\) and readl\(\), read\(\) and read\(\)](#) — [Read data from a file using a handle](#)
[remove\(\)](#) — [Delete a file](#)
[rename\(\)](#) — [Change the name of a file](#)
[tell\(\) and telll\(\), tell\(\) and tell\(\)](#) — [Report the current position in a file](#)
[umask\(\) and umaskl\(\), umask\(\) and umask\(\)](#) — [Set global file permission mask](#)
[unlink\(\) and unlinkl\(\), unlink\(\) and unlink\(\)](#) — [Delete a file](#)
[write\(\) and writel\(\), write\(\) and write\(\)](#) — [Write data to a file, unbuffered](#)

Structure

`struct finddata_t` — [Attributes of a file](#)

Type

`time_t` — [Representation of a date and time](#)

limits.h

Header file `limits.h` defines constants that constrain the representation of numeric values. The actual values of these constants vary from system to system.

NOTE `limits.h` is an ANSI-defined header file.

Constants

[CHAR *](#) — [Minimum and maximum values for char types](#)

[INT *](#) — [Minimum and maximum values for int types](#)

[LONG *](#) — [Minimum and maximum values for signed long types](#)

[MB_LEN_MAX](#) — [Maximum number of bytes in a multibyte character](#)

[SCHAR *](#) — [Minimum and maximum values for signed char types](#)

[SHRT *](#) — [Minimum and maximum values for short types](#)

[U* MAX](#) — [Maximum values for unsigned types](#)

locale.h

Header file `locale.h` defines functions, types, and macros pertaining to geographic localization of numeric formats.

NOTE `locale.h` is an ANSI-defined header file.

Functions

[localeconv\(\)](#) — [Query current locale for numeric format](#)

[setlocale\(\)](#) — [Set the current locale](#)

Structure

[struct lconv](#) — [Locale-specific numeric representation](#)

Constants

[LC *](#) — [Arguments to setlocale\(\)](#)

[NULL](#) — [The null pointer](#)

locking.h

Header file `locking.h` defines constants needed by the file-locking function, `locking()`, to determine locking, unlocking, and locking with retries.

NOTE `locking.h` is available only for DOS and Windows NT targets.

Constants

[LK_* and _LK_*, LK_* and _LK_*](#) — [File-locking mode](#)

malloc.h

Header file `malloc.h` defines memory-allocation functions. Might be operating-system dependent.

Functions

[alloca\(\) and alloca\(\)](#) — [Allocate memory on the stack](#)

[calloc\(\)](#) — [Dynamically allocate zero-initialized storage for objects](#)

[free\(\)](#) — [Release storage allocated by calloc\(\), malloc\(\), or realloc\(\)](#)

[heapchk\(\)](#) — [Check heap for consistency](#)

[heapset\(\)](#) — [Fill free memory locations in the heap](#)

[heapwalk\(\)](#) — [Walk through the heap](#)

[malloc\(\)](#) — [Dynamically allocate uninitialized storage](#)

[realloc\(\)](#) — [Reallocate storage allocated by calloc\(\) or malloc\(\)](#)

Structure

struct [HEAPINFO](#) — [Information about the heap](#)

Constants

[HEAP*](#) — [State of the heap](#)

math.h

Header file `math.h` declares mathematical functions and defines mathematical constants.

NOTE `math.h` is an ANSI-defined header file.

Functions

[acos\(\)](#) — [Arc cosine](#)

[acosh\(\) and acosh\(\), acosh\(\) and acosh\(\)](#) — [Hyperbolic arc cosine](#)

[asin\(\)](#) — [Arc sine](#)

[asinh\(\) and asinh\(\), asinh\(\) and asinh\(\)](#) — [Hyperbolic arc sine](#)

[atan\(\)](#) — [Arc tangent](#)

[atan2\(\)](#) — [Arc tangent of the angle defined by a point](#)

[atanh\(\) and atanh\(\), atanh\(\) and atanh\(\)](#) — [Hyperbolic arc tangent](#)

[atof\(\)](#) — [Convert a prefix of a string to floating point](#)

[cabs\(\) and cabs\(\), cabs\(\) and cabs\(\)](#) — [Compute complex absolute value](#)

[ceil\(\)](#) — [Calculate smallest integer greater than or equal to a value](#)

[cos\(\)](#) — [Cosine](#)

[coshf\(\)](#) — [Hyperbolic cosine using single-precision floating point](#)

[dieetombsin\(\) and dieetombsin\(\), dieetombsin\(\) and dieetombsin\(\)](#) — [Convert IEEE double to Microsoft double](#)

[dmsbintoieee\(\)](#) and [_dmsbintoieee\(\)](#), [dmsbintoieee\(\)](#) and [_dmsbintoieee\(\)](#) — Convert Microsoft double to IEEE double

[exp\(\)](#) — Calculate exponential (e to the power x)

[fabs\(\)](#) — Calculate absolute value of a floating-point number using double precision

[fieetomsbin\(\)](#) and [_fieetomsbin\(\)](#), [fieetomsbin\(\)](#) and [_fieetomsbin\(\)](#) — Convert IEEE float to Microsoft format

[floor\(\)](#) — Calculate largest integer not greater than a value

[fmod\(\)](#) — Return floating-point remainder

[fmsbintoieee\(\)](#) and [_fmsbintoieee\(\)](#), [fmsbintoieee\(\)](#) and [_fmsbintoieee\(\)](#) — Convert Microsoft float to IEEE float

[frexp\(\)](#) — Break a double into fraction and exponent

[hypot\(\)](#) and [_hypot\(\)](#), [hypot\(\)](#) and [_hypot\(\)](#) — Hypotenuse of a triangle

[j0\(\)](#), [j1\(\)](#), [jn\(\)](#) and [_j0\(\)](#), [_j1\(\)](#), [_jn\(\)](#), [j0\(\)](#), [j1\(\)](#), [jn\(\)](#) and [_j0\(\)](#), [_j1\(\)](#), [_jn\(\)](#) — Bessel function of the first kind

[ldexp\(\)](#) — Multiply by a power of 2

[log\(\)](#) — Calculate natural logarithm (base e)

[log10\(\)](#) — Calculate common logarithm (base 10)

[log2\(\)](#) and [_log2\(\)](#), [log2\(\)](#) and [_log2\(\)](#) — Calculate logarithm (base 2)

[matherr\(\)](#), [matherrx\(\)](#) and [_matherr\(\)](#), [_matherrx\(\)](#), [matherr\(\)](#), [matherrx\(\)](#) and [_matherr\(\)](#), [_matherrx\(\)](#) — Provide standard error reporting for math functions

[modf\(\)](#) — Break a double into integer and fractional parts

[pow\(\)](#) — Raise a double to a power

[set_matherr\(\)](#) — User defined math error function

[sin\(\)](#) — Sine

[sinh\(\)](#) — Hyperbolic sine

[sqrt\(\)](#) — Square root

[tan\(\)](#) — Tangent

[tanh\(\)](#) — Hyperbolic tangent

[y0\(\)](#), [y1\(\)](#), [yn\(\)](#) and [_y0\(\)](#), [_y1\(\)](#), [_yn\(\)](#), [y0\(\)](#), [y1\(\)](#), [yn\(\)](#) and [_y0\(\)](#), [_y1\(\)](#), [_yn\(\)](#) — Bessel function of the second kind

Structures

[struct complex](#) — A complex number

[struct exception](#) — Nature of a math error

Constants

[E](#) — Natural logarithm base e

[EDOM](#) — Mathematical domain error

[ERANGE](#) — Mathematical range error

[HUGE_VAL](#) — Largest positive floating-point value

[PI](#) — Value of pi

Using Global Variable `errno` with Mathematical Functions

Several mathematical functions reference the `int` variable `errno`, which is declared in `errno.h` and is set to 0 (zero) at program start-up. Some library functions set `errno` to a positive integer error code when an error occurs during their execution. No library function ever sets `errno` to 0 (zero).

- When an argument to a function is outside the domain over which the function is (mathematically) defined, `errno` is set to **EDOM**.
- When the result of a function is too large in magnitude to fit in a **double**, `errno` is set to **ERANGE**.

To reference `errno` from a program, the header file `errno.h` must be included. For more information about global variable `errno`, see [Constants and Globals](#) on page 319.

NOTE Using `errno` can affect the reentrancy of mathematical functions. See [Reentrancy](#) on page 25.

memory.h

Header file `memory.h` defines several functions for moving and comparing portions of memory.

Functions

[memccpy\(\)](#) and [memccpy\(\)](#), [memcpy\(\)](#) and [memccpy\(\)](#) — Copy memory until count or character

[memchr\(\)](#) — Find a character in an area of memory

[memcmp\(\)](#) — Compare two areas of memory

[memcpy\(\)](#) — Copy from one place in memory to another

[memicmp\(\)](#) and [memicmp\(\)](#), [memicmp\(\)](#) and [memicmp\(\)](#) — Compare bytes ignoring case

[memset\(\)](#) — Duplicate a character across an area of memory

_na.h

Header file `na.h` defines non-ANSI names and functions. Definitions in this header file vary greatly from system to system. See your installation for specifics.

process.h

Header file `process.h` defines the functions and constants needed to control the active processes running in the computer system.

NOTE `process.h` is only for DOS and Windows NT targets.

Functions

[abort\(\)](#) — [Terminate a program abnormally](#)

[exit\(\)](#) — [Terminate a program normally](#)

[getpid\(\) and _getpid\(\), getpid\(\) and _getpid\(\)](#) — [Get unique ID of the calling process](#)

[spawn*\(\) and _spawn*\(\), spawn*\(\) and _spawn*\(\)](#) — [Execute a child process](#)

[system\(\)](#) — [Pass a command to the operating system](#)

Constants

[P_*](#) — [Determine mode of child and behavior of parent](#)

[threadid](#) — [ID of currently running thread](#)

search.h

Header file search.h defines sorting and searching functions.

Functions

[bsearch\(\)](#) — [Perform a binary search](#)

[lfind\(\) and _lfind\(\), lfind\(\) and _lfind\(\)](#) — [Search a linear list for a matching value, without modifying the list](#)

[lsearch\(\) and _lsearch\(\), lsearch\(\) and _lsearch\(\)](#) — [Search a linear list for a matching value, modifying if necessary](#)

[qsort\(\)](#) — [Perform a quicksort](#)

Type

[size_t](#) — [Type of the result of sizeof\(\)](#)

setjmp.h

Header file setjmp.h defines a type and two functions for setting up and executing non-local jumps.

Note setjmp.h is an ANSI-defined header file.

Functions

[longjmp\(\)](#) — [Execute a non-local jump](#)

[setjmp\(\)](#) — [Save a reference to the current calling environment for a subsequent non-local jump](#)

Type

[jmp_buf](#) — [Information to restore an environment](#)

share.h

Header file share.h defines the constants needed to specify the allowable sharing modes for the DOS [open\(\)](#) and [_fsopen\(\)](#) functions.

Note share.h is available only for DOS and Windows NT targets.

Constants

[SH](#)* — [File-sharing flags](#)

signal.h

Header file `signal.h` declares functions and defines several macros for handling signals. For a complete list of signal definitions (including floating-point exceptions) and arguments to the signal-handling functions, see the contents of file `signal.h` in your MetaWare C installation. Signal handling is system dependent. More signal codes might be defined for specific systems.

NOTE `signal.h` is an ANSI-defined header file.

Functions

[raise\(\)](#) — [Raise a signal](#)

[signal\(\)](#) — [Set up a signal handler](#)

Type

[sig_atomic_t](#) — [Integral type; atomic entity](#)

Constants

[SIG_DFL](#) and [SIG_ERR](#) and [SIG_IGN](#), [SIG_DFL](#) and [SIG_ERR](#) and [SIG_IGN](#), [SIG_DFL](#) and [SIG_ERR](#) and [SIG_IGN](#) — Macros; arguments to `signal()`

[SIG*](#) — Arguments to `signal()` and `raise()`

sizet.h

Header file `sizet.h` defines type `size_t`, the type of the result of the `sizeof()` operator.

stat.h

Header file `stat.h` defines the [stat\(\) and _stat\(\)](#) function and the types and constants it needs.

NOTE Sometimes `stat.h` is located in the `sys` subdirectory, and can be specified as `sys\stat.h`.

Functions

[chmod\(\) and _chmod\(\)](#), [chmod\(\) and _chmod\(\)](#) — [Alter the permission setting for a file](#)

[mkdir\(\) and _mkdir\(\)](#), [mkdir\(\) and _mkdir\(\)](#) — [Create a new directory](#)

[stat\(\) and _stat\(\)](#), [stat\(\) and _stat\(\)](#) — [Get information about a file or directory](#)

[umask\(\) and _umask\(\)](#), [umask\(\) and _umask\(\)](#) — [Set global file permission mask](#)

Constants

[S_I*](#) — [File permissions](#)

Structure

`struct stat` — [Information about a file](#)

Type[time_t — Representation of a date and time](#)

stdarg.h

Header file `stdarg.h` defines a type and several macros for varying the numbers and types of arguments to a given function. The macros assist in advancing through the actual sequence of arguments in order, first to last, one by one. All definitions in this header file conform to the ANSI C Standard. However, `stdarg.h` includes `_stdarg.h`, which defines Extra-ANSI macros.

Note `stdarg.h` is an ANSI-defined header file.

Functions[va_arg\(\) — Get the next argument in a variable series of arguments](#)[va_end\(\) — Terminate va_list\(\) processing](#)[va_start\(\) — Initialize an object of type va_list](#)**Type**[va_list — Information needed by var-arg macros](#)**Using Variable-Argument Macros**

To use the var-arg (variable-argument) macros, a function must have at least one declared argument whose name and type are known. The following functions meet the requirements:

```
void f(x),      ff(x,y,z,...);
int fff(x,y), *ffff(a,...);
```

These functions do not meet the requirements:

```
char *g(),     gg(...);
```

The right-most named argument is used as a starting place for advancing through the sequence.

stddef.h

Header file `stddef.h` defines useful macros, some of which are also defined in other header files.

Note `stddef.h` is an ANSI-defined header file.

Functions[offsetof\(\) — Determine offset of a structure member](#)**Types**[ptrdiff_t — Integral type; difference of two pointers](#)[size_t — Type of the result of sizeof\(\)](#)[wchar_t — Integral type; extended-character-set values](#)**Constants**[NULL — The null pointer](#)

Global Variable

[threadid](#) — ID of currently running thread

stdio.h

Header file stdio.h declares functions, macros, a type, and a variable used for input and output (I/O). For more information about standard I/O, see [Input and Output](#) on page 385.

NOTE stdio.h is an ANSI-defined header file.

Functions

[bprintf\(\)](#) — Print to a string of given size

[clearerr\(\)](#) — Clear end-of-file and error flags

[fclose\(\)](#) — Close a file

[fcloseall\(\)](#) and [_fcloseall\(\)](#), [fcloseall\(\)](#) and [_fcloseall\(\)](#) — Close all files

[fdopen\(\)](#) and [_fdopen\(\)](#), [fdopen\(\)](#) and [_fdopen\(\)](#) — Associate a stream with an open handle

[feof\(\)](#) — Test for end-of-file

[ferror\(\)](#) — Test for a read or write error on a file

[fflush\(\)](#) — Flush a file's buffer

[fgetc\(\)](#) — Read (get) a character from a file

[fgetchar\(\)](#) and [_fgetchar\(\)](#), [fgetchar\(\)](#) and [_fgetchar\(\)](#) — Get a character from standard input

[fgetpos\(\)](#) — Determine file position

[fgets\(\)](#) — Read a line of text (string) from a file

[filelength\(\)](#) and [_filelength\(\)](#), [filelength\(\)](#) and [_filelength\(\)](#) — Determine length of an open file

[fileno\(\)](#) and [_fileno\(\)](#), [fileno\(\)](#) and [_fileno\(\)](#) — Get file handle currently associated with a stream

[flushall\(\)](#) and [_flushall\(\)](#), [flushall\(\)](#) and [_flushall\(\)](#) — Clear input buffers and flush output buffers

[fopen\(\)](#) — Open a file

[fprintf\(\)](#) — Print to a file

[fputc\(\)](#) — Write a character to a file

[fputchar\(\)](#) and [_fputchar\(\)](#), [fputchar\(\)](#) and [_fputchar\(\)](#) — Write a character to standard output

[fputs\(\)](#) — Write a string to a file

[fread\(\)](#) — Read from a file

[freopen\(\)](#) — Open a file using an existing FILE variable

[fscanf\(\)](#) — Read values from a file

[fseek\(\)](#) — Move file pointer to new location in file

[fsetpos\(\)](#) — Set file position

[ftell\(\)](#) — Determine current value of a file pointer

[fwrite\(\)](#) — Write to a file
[getc\(\)](#) — Get a character from a file
[getchar\(\)](#) — Get a character from standard input
[gets\(\)](#) — Read a line of text (string) from standard input
[getw\(\) and getw\(\), getw\(\) and getw\(\)](#) — Get an integer from a file
[mktemp\(\) and mktemp\(\), mktemp\(\) and mktemp\(\)](#) — Create a unique file name (does not create or open files)
[perror\(\)](#) — Print an error message
[printf\(\)](#) — Print to `stdout`
[putc\(\)](#) — Write a character to a file
[putchar\(\)](#) — Write a character to standard output
[puts\(\)](#) — Write a string to standard output
[putw\(\) and putw\(\), putw\(\) and putw\(\)](#) — Write an integer to a file
[remove\(\)](#) — Delete a file
[rename\(\)](#) — Change the name of a file
[rewind\(\)](#) — Seek to the beginning of a file
[scanf\(\)](#) — Read values from `stdin`
[setbuf\(\)](#) — Specify a buffer for a stream
[setmode\(\)](#) — Set stream mode (`_fmode`) to text or binary
[setvbuf\(\)](#) — Control file buffering
[sprintf\(\)](#) — Print to a string
[sscanf\(\)](#) — Read values from a string
[tell\(\) and tell\(\), tell\(\) and tell\(\)](#) — Report the current position in a file
[tempnam\(\)](#) — Generate a temporary file name in a directory
[tmpfile\(\) and tmpfile\(\)](#) — Create a temporary file
[tmpnam\(\)](#) — Generate a string to be used as a temporary file name
[ungetc\(\)](#) — Push a character back into an input stream
[unlink\(\) and unlink\(\), unlink\(\) and unlink\(\)](#) — Delete a file
[vprintf\(\)](#) — Print to a string of given size using var-arg macros
[vfprintf\(\)](#) — Print to a file using var-arg macros
[vfscanf\(\) and vfscanf\(\), vfscanf\(\) and vfscanf\(\)](#) — Read values from a file using var-arg macros
[vprintf\(\)](#) — Print to `stdout` using var-arg macros
[vscanf\(\) and vscanf\(\), vscanf\(\) and vscanf\(\)](#) — Read values from `stdin` using var-arg macros
[vsprintf\(\)](#) — Print to a string using var-arg macros

[vsscanf\(\) and _vsscanf\(\), vsscanf\(\) and _vsscanf\(\)](#) — Read values from a string using var-arg macros

Constants

[BUFSIZ](#) — Size of a buffer used by setbuf()

[EOF](#) — End-of-file indicator

[FILENAME_MAX](#) — Maximum number of characters in a file name

[FOPEN_MAX](#) — Maximum number of open files

[IOWR and IOWR and IOWR, IOWR and IOWR and IOWR, IOWR and IOWR and IOWR — Buffering-mode arguments to setvbuf\(\)](#)

[L_tmpnam](#) — Size of a temporary file name

[NULL](#) — The null pointer

[SEEK_CUR and SEEK_END and SEEK_SET, SEEK_CUR and SEEK_END and SEEK_SET, SEEK_CUR and SEEK_END and SEEK_SET — Macros used to specify position in file](#)

[TMP_MAX](#) — Minimum number of unique file names generated by tmpnam()

Types

[fpos_t](#) — Unique file-position type

[size_t](#) — Type of the result of sizeof()

Structures

[typedef struct FILE](#) — Information for controlling a stream

[typedef struct ldiv_t](#) — Result type of function ldiv()

Global Variables

[stderr and stdin and stdout, stderr and stdin and stdout, stderr and stdin and stdout](#) — Standard error, input, and output streams

stdlib.h

Header file stdlib.h declares several types, macros, and functions of general utility. Among these are functions that perform the following categories of tasks:

- converting strings to numeric values
- communicating with the host environment
- managing memory
- generating pseudo-random integers

NOTE stdlib.h is an ANSI-defined header file.

Functions

[abort\(\)](#) — Terminate a program abnormally

[abs\(\)](#) — Calculate the absolute value of an integer

[atexit\(\)](#) — Register a function for execution at program termination

[atof\(\)](#) — Convert a prefix of a string to floating point

[atoi\(\)](#) — Convert a prefix of a string to an int
[atol\(\)](#) — Convert a prefix of a string to a long int
[atold\(\)](#) — Convert a prefix of a string to a long double
[bsearch\(\)](#) — Perform a binary search
[calloc\(\)](#) — Dynamically allocate zero-initialized storage for objects
[crotl\(\) and _crotr\(\), _crotl\(\) and _crotr\(\)](#) — Rotate a char left or right
[div\(\)](#) — Perform integer division with remainder
[ecvt\(\) and _ecvt\(\), ecvt\(\) and _ecvt\(\)](#) — Convert a floating-point number to a character string
[exit\(\)](#) — Terminate a program normally
[_exit\(\)](#) — Terminate a program immediately
[fcvt\(\) and _fcvt\(\), fcvt\(\) and _fcvt\(\)](#) — Convert a floating-point number to a character string
[free\(\)](#) — Release storage allocated by calloc(), malloc(), or realloc()
[fullpath\(\)](#) — Convert relative pathname to absolute pathname
[gcvt\(\) and _gcvt\(\), gcvt\(\) and _gcvt\(\)](#) — Convert a floating-point number to a character string
[getenv\(\)](#) — Get the value of an environment variable
[initcopy\(\)](#) — Copy initial values from ROM to RAM
[itoa\(\) and _itoa\(\), itoa\(\) and _itoa\(\)](#) — Convert an int value to an ASCII string
[labs\(\)](#) — Calculate the absolute value of a long int
[ldecvt\(\)](#) — Convert a long double to a character string
[ldfcvt\(\)](#) — Convert a long double to a character string
[ldiv\(\)](#) — Perform long int division with remainder
[lrotl\(\) and _lrotr\(\), lrotl\(\) and _lrotr\(\)](#) — Rotate a long int left or right
[ltoa\(\) and _ltoa\(\), ltoa\(\) and _ltoa\(\)](#) — Convert a long int value to an ASCII string
[makepath\(\)](#) — Create a path string from components
[malloc\(\)](#) — Dynamically allocate uninitialized storage
[mblen\(\)](#) — Determine length of a multibyte character
[mbstowcs\(\)](#) — Convert a multibyte string to wchar_t array
[mbtowc\(\)](#) — Convert a multibyte character to wchar_t code
[msize\(\)](#) — Get the size of a memory block in the heap
[perror\(\)](#) — Print an error message
[putenv\(\) and _putenv\(\), putenv\(\) and _putenv\(\)](#) — Add or modify an environment variable
[qsort\(\)](#) — Perform a quicksort
[rand\(\)](#) — Generate a pseudo-random number
[realloc\(\)](#) — Reallocate storage allocated by calloc() or malloc()

[rotl\(\) and _rotl\(\)](#), [_rotl\(\) and _rotr\(\)](#) — Rotate an int left or right
[searchenv\(\)](#) — Search for a file using an environment variable
[searchstr\(\)](#) — Search for a file in a list of directories
[sleep\(\)](#) — Temporarily suspend program execution
[splitpath\(\)](#) — Split a pathname into its components
[rand\(\)](#) — Seed the pseudo-random number generator
[srotl\(\) and _srotl\(\)](#), [_srotl\(\) and _srotr\(\)](#) — Rotate a short int left or right
[strtod\(\)](#) — Convert a string to a double
[strtol\(\)](#) — Convert a string to a long
[strtoul\(\)](#) — Convert a string to an unsigned long
[swab\(\) and _swab\(\)](#), [swab\(\) and _swab\(\)](#) — Swap bytes in a word
[system\(\)](#) — Pass a command to the operating system
[tolower\(\)](#) — Convert to lowercase
[toupper\(\)](#) — Convert to uppercase
[ultoa\(\) and _ultoa\(\)](#), [ultoa\(\) and _ultoa\(\)](#) — Convert an unsigned long to an ASCII string
[utoa\(\) and _utoa\(\)](#), [utoa\(\) and _utoa\(\)](#) — Convert an unsigned int to an ASCII string
[wcstombs\(\)](#) — Convert a wchar_t array to a multibyte string
[wctomb\(\)](#) — Convert a wchar_t code to a multibyte character

Constants

[EXIT_FAILURE and EXIT_SUCCESS](#), [EXIT_FAILURE and EXIT_SUCCESS](#) — Arguments to `exit()`

[MAX_*](#) — Maximum length of several pathname components

[MB_CUR_MAX](#) — Maximum number of bytes for a character in current locale

[NULL](#) — The null pointer

[RAND_MAX](#) — Maximum random number

Structures

[typedef struct div_t](#) — Result type of function `div()`

[typedef struct ldiv_t](#) — Result type of function `ldiv()`

Types

[wchar_t](#) — Integral type; extended-character-set values

string.h

Header file `string.h` defines macros and types and declares functions for manipulating strings and arbitrary areas of memory.

The string functions (**str***()) take strings as arguments. The functions designed to manipulate arbitrary areas of memory (**mem***()) take (**void ***) pointers to any type of object as parameters. Many of these functions are also provided as macros.

The copy routines whose names begin with **_r** (for *reverse*) copy from high addresses to low addresses; the other copy routines copy from low addresses to high addresses.

Note `string.h` is an ANSI-defined header file.

Functions

[**memccpy\(\)**](#) and [**_memccpy\(\)**](#), [**memccpy\(\)**](#) and [**_memccpy\(\)**](#) — Copy memory until count or character

[**memchr\(\)**](#) — Find a character in an area of memory

[**memcmp\(\)**](#) — Compare two areas of memory

[**memcpy\(\)**](#) — Copy from one place in memory to another

[**memicmp\(\)**](#) and [**_memicmp\(\)**](#), [**memicmp\(\)**](#) and [**_memicmp\(\)**](#) — Compare bytes ignoring case

[**memmove\(\)**](#) — Copy memory nondestructively

[**memset\(\)**](#) — Duplicate a character across an area of memory

[**rmemcpy\(\)**](#) — Copy from one place in memory to another

[**rstrcpy\(\)**](#) — Copy one string onto another

[**rstrncpy\(\)**](#) — Copy a number of characters from one string into another

[**streat\(\)**](#) — Concatenate two strings

[**streats\(\)**](#) — Concatenate multiple strings up to a limited number of characters

[**strchr\(\)**](#) — Find the first occurrence of a character in a string

[**strcmp\(\)**](#) — Compare one string to another

[**stremp\(\)**](#) and [**_stremp\(\)**](#), [**stremp\(\)**](#) and [**_stremp\(\)**](#) — Compare strings, ignoring case

[**strcoll\(\)**](#) — Compare strings based on current locale

[**strcpy\(\)**](#) — Copy the characters of one string into another

[**strcspn\(\)**](#) — Determine the length of the prefix of a string not containing any characters from another string

[**strdup\(\)**](#) and [**_strdup\(\)**](#), [**strdup\(\)**](#) and [**_strdup\(\)**](#) — Make a copy of a string in the heap

[**strerror\(\)**](#) and [**_strerror\(\)**](#), [**strerror\(\)**](#) and [**_strerror\(\)**](#) — Map error code; append error-message string to user message

[**stricmp\(\)**](#) and [**_stricmp\(\)**](#), [**stricmp\(\)**](#) and [**_stricmp\(\)**](#) — Compare strings, ignoring case

[**strlen\(\)**](#) — Determine the length of a string

[**strlwr\(\)**](#) and [**_strlwr\(\)**](#), [**strlwr\(\)**](#) and [**_strlwr\(\)**](#) — Make string all lowercase

[**strncat\(\)**](#) and [**_strncat\(\)**](#), [**strncat\(\)**](#) and [**_strncat\(\)**](#) — Append characters of one string to another, up to some limit n

[strcmp\(\)](#) — Compare characters of one string to characters of another
[strcpy\(\)](#) — Copy a number of characters from one string into another
[strnicmp\(\) and _strnicmp\(\), strnicmp\(\) and _strnicmp\(\)](#) — Compare strings, ignoring case
[strnset\(\) and _strnset\(\), strnset\(\) and _strnset\(\)](#) — Fill part or all of a string with any character
[strpbrk\(\)](#) — Find the first occurrence of any character of one string in another
[strchr\(\)](#) — Find the last occurrence of a character in a string
[strrev\(\) and _strrev\(\), strrev\(\) and _strrev\(\)](#) — Reverse the order of characters in a string
[strset\(\) and _strset\(\), strset\(\) and _strset\(\)](#) — Fill a string with any character
[strspn\(\)](#) — Determine the length of the prefix of a string composed entirely of characters from another string
[strstr\(\)](#) — Find the first occurrence of one string within another
[strtok\(\)](#) — Divide a string into tokens
[strupr\(\) and _strupr\(\), strupr\(\) and _strupr\(\)](#) — Make string all uppercase
[strxfrm\(\)](#) — Transform string to locale-independent form

Constants
[MAXSTRING](#) — Maximum length of a string

Types
[size_t](#) — Type of the result of sizeof()

time.h

Header file `time.h` declares types and functions and defines a macro for manipulating representations of time.

NOTE `time.h` is an ANSI-defined header file.

Functions

[asctime\(\) and _asctime\(\), asctime\(\) and _asctime\(\)](#) — Convert a struct tm to printable form
[clock\(\)](#) — Report elapsed processor time
[ctime\(\) and _ctime\(\), ctime\(\) and _ctime\(\)](#) — Convert a time_t value to printable form
[difftime\(\)](#) — Calculate difference between two times
[gmtime\(\) and _gmtime\(\), gmtime\(\) and _gmtime\(\)](#) — Convert a time_t value to a struct tm, adjusted to UTC
[localtime\(\) and _localtime\(\), localtime\(\) and _localtime\(\)](#) — Convert a time_t value to a struct tm
[mktime\(\)](#) — Convert a struct tm to a time_t value
[strdate\(\)](#) — Convert the current date to a string
[strftime\(\)](#) — Format time and date string

[strftime\(\)](#) — Convert current time of day to a string

[time\(\)](#) — Get the current time and date

[tzset\(\) and _tzset\(\), tzset\(\) and _tzset\(\)](#) — Set local time-zone values

Structure

[struct tm](#) — Components of a date and time

Constants

[CLOCKS_PER_SEC](#) — Clock ticks converted to seconds

[NULL](#) — The null pointer

Types

[clock_t](#) — Representation of elapsed processor time

[size_t](#) — Type of the result of sizeof()

[time_t](#) — Representation of a date and time

Global variables

[daylight, _daylight](#) — Pacific Daylight Time

[timezone, _timezone](#) — Time zone; hours from UTC, expressed in seconds

timeb.h

Header file `timeb.h` provides definitions needed by the `ftime()` function.

Note `timeb.h` is available only for DOS and Windows NT targets.

Functions

[ftime\(\) and _ftime\(\), ftime\(\) and _ftime\(\)](#) — Get current time values

Structures

[struct timeb](#) — Time-zone information from function `ftime()`

Types

[time_t](#) — Representation of a date and time

types.h

Header file `types.h` defines types returned by time and file-status system calls.

utime.h

Header file `utime.h` provides declarations needed by the `utime()` function.

Note `utime.h` is available only for DOS and Windows NT targets.

Functions

[utime\(\) and _utime\(\), utime\(\) and _utime\(\)](#) — Update a file's date/time stamp

Structures

struct utimbuf — [Information about file access and modification times](#)

Types

time_t — [Representation of a date and time](#)

varargs.h

Header file varargs.h is an alternate name for stdarg.h, supplied for compatibility with systems that do not use the ANSI C Standard name. See [stdarg.h](#) on page 39.

Chapter 4 — Function Reference

In This Chapter

- [Function Listing Format](#)
- [Function Listings](#)

Function Listing Format

This is the general format of an entry for a function or macro:

item_name()

What **item_name()** does

Targets

Shows target platforms ('x') on which **item_name()** is supported:

Emb	UNIX	DOS	NT
x	x	x	x

- **Emb** indicates Embedded systems (non-Intel), such as ARC, ARM, PowerPC, and so on. An entry of "ARC" means that the function is available for ARC targets only.
- **UNIX** indicates most UNIX systems: Vr.4, BSD, HPUX, and so on.
- **DOS** indicates 32-bit extended-DOS, Windows 3.x, or Embedded extended-DOS ("xDOS").
- **NT** indicates Windows NT or Windows 95 for Intel.

ANSI

Specifies level of ANSI C Standard compliance of **item_name()**:

- ANSI compliant (ANSI Standard C function, type, or macro)
- C99 (ISO/ANSI C99 function)
- Extra-ANSI (not required by the ANSI C Standard; follows ANSI naming conventions)
- Non-ANSI (not required by the ANSI C Standard; violates ANSI naming conventions).

Reentrancy

Specifies whether **item_name()** is Reentrant or Not-Reentrant. If reentrancy does not apply, no label appears.

Synopsis

```
#include <stdlib.h>      /* Required */  
void item(void);
```

Provides a prototype for **item_name()**. If /* Required */ appears, the specified header file(s) must be included.

Description

Describes **item_name()**. If **item_name()** is a function or macro, this section describes the arguments; if **item_name()** is a type or a constant, this section explains the context in which it is used.

System issues

Specifies additional constraints relevant to certain target platforms.

Return Value

Specifies return values if **item_name()** is a function.

See Also

Lists other items functionally or conceptually related to **item_name()**.

Example

Specifies the name and directory location of an example program, provided on the distribution, that illustrates the use of **item_name()**.

Function Listings

NOTE In the following listings, names beginning with or containing an underscore (_) are alphabetized as though the underscore were not present.

abort()

Terminate a program abnormally

abort() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>
void abort(void);
```

Description

abort() causes the program to terminate, unless **signal()** is used. **abort()** does not close any open streams or remove any temporary files.

Return Value

abort() returns -1 to the program's calling environment by calling **raise(SIGABRT)**.

See Also

[exit\(\)](#) — [Terminate a program normally](#)

[raise\(\)](#) — [Raise a signal](#)

[signal\(\)](#) — [Set up a signal handler](#)

Example

See the file `abort.c` in the `install_dir/docs/examples/` directory on your distribution.

abs()

Calculate the absolute value of an integer

Targets

`_abs()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
int abs(int x);
```

Description

`abs()` computes the absolute value of x . `abs()` might be inlined when toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

`abs()` returns the absolute value of x .

CAUTION Given arguments outside the range of `int`, `abs()` returns an undefined result.

Surprises

`abs(INT_MIN)` returns `INT_MIN`, because there is no corresponding positive integer.

See Also

[cabs\(\) and _cabs\(\) — Compute complex absolute value](#)

[fabs\(\) — Calculate absolute value of a floating-point number using double precision](#)

[labs\(\) — Calculate the absolute value of a long int](#)

Constants [INT_* — Minimum and maximum values for int types](#)

Example

See the file `abs.c` in the `install_dir/docs/examples/` directory on your distribution.

_abs()

Calculate the absolute value of any arithmetic type

Targets

`_abs()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis`_abs(x)`**Description**`_abs()` is a compiler intrinsic that calculates the absolute value of x , where x is any arithmetic type.

Note Unless you specify command-line options **-Hansi** or **-Hpcc**, `_abs()` replaces the functions `abs()`, `labs()`, and `fabs()`.

Return Value`_abs()` returns the absolute value of x . The returned value is of the same type as x .**See Also**[abs\(\)](#) — Calculate the absolute value of an integer[fabs\(\)](#) — Calculate absolute value of a floating-point number using double precision[labs\(\)](#) — Calculate the absolute value of a long int

access() and _access()

Test access rights of a file

Targets`access()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

`_access()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI`access()` is Non-ANSI; `_access()` is Extra-ANSI.**Reentrancy**

Both functions are Not-Reentrant.

Synopsis

```
/* Required for DOS and Windows targets: */
#include <iо.h>
/* Required for all other targets: */
#include <unistd.h>

int access(char *path, int mode);
int _access(char *path, int mode);
```

Description`access()` and `_access()` determine if a specified *path* exists and can be accessed in *mode*. These are the possible values for *mode*:

- 0: Test for existence only.
- 2: Test for write permission.
- 4: Test for read permission.
- 6: Test for read and write permission.

Return Value

If *path* refers to a file, **access()** and **_access()** return 0 (zero) if the file exists and can be accessed in *mode*.

If *path* refers to a directory, **access()** and **_access()** test for existence only, returning 0 (zero) if the directory exists.

If *path* does not exist, or is not accessible in the specified *mode*, **access()** and **_access()** return -1 and set the global variable **errno** to one of the following values:

EACCES — Access error; permission denied

ENOENT — File or directory not found; pathname element does not exist

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

On DOS, the *mode* argument is relevant only when *path* refers to a file, because directories always have read and write access on DOS.

On Windows NT, **_access()** tests for existence and for the actual access requested.

See Also

[File-Related Functions](#) on page 429

Global variable **errno** — An int used to record error numbers

Example

See the file *access.c* in the *install_dir/docs/examples/* directory on your distribution.

acos()

Arc cosine

Targets

acos() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
double acos(double x);
```

Description

acos() computes the principal value of the arc cosine of x . The value for x must be in the domain [-1.0, 1.0]. **acos()** might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

acos() returns the arc cosine of x (assuming x is in the domain [-1.0, 1.0]). The return value is in the range $[0, \pi]$.

When x is outside the domain, **acos()** returns 0 (zero) and sets global variable `errno` to [EDOM](#) ([Mathematical domain error](#)).

See Also

[**acosf\(\)** — Arc cosine using single-precision floating point](#)

[**asin\(\)** — Arc sine](#)

[**atan\(\)** — Arc tangent](#)

[**cos\(\)** — Cosine](#)

[**cosf\(\)** — Cosine using single-precision floating point](#)

Global variable `errno` — [An int used to record error numbers](#)

[Transcendental Math Functions](#) on page 424

acosf()

Arc cosine using single-precision floating point

Targets

acosf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
float acosf(float x);
```

Description

acosf() computes the principal value of the arc cosine of x . The value for x must be in the domain [-1.0, 1.0]. **acosf()** might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

acosf() returns the arc cosine of x (assuming x is in the domain [-1.0, 1.0]). The return value is in the range $[0, \pi]$.

When x is outside the domain, **acosf()** returns 0 (zero) and sets global variable `errno` to [EDOM](#) ([Mathematical domain error](#)).

See Also

[acos\(\)](#) — Arc cosine

[asin\(\)](#) — Arc sine

[asinf\(\)](#) — Arc sine using single-precision floating point

[asinf\(\)](#) — Arc sine using single-precision floating point

[atan\(\)](#) — Arc tangent

[atanf\(\)](#) — Arc tangent using single-precision floating point

[cos\(\)](#) — Cosine

[cosf\(\)](#) — Cosine using single-precision floating point

Global variable [errno](#) — An int used to record error numbers

[Transcendental Math Functions](#) on page 424

acosh() and _acosh()

Hyperbolic arc cosine

Targets

acosh() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_acosh() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

acosh() is Non-ANSI; **_acosh()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h>           /* Required */  
double _acosh(double x);
```

Description

acosh() computes the inverse hyperbolic cosine of x , using the following calculation:

$$\text{acosh}(x) = \log(x + \sqrt{x^2 - 1})$$

Return Value

acosh() returns the inverse hyperbolic cosine of x (assuming $x \geq 1$).

If the value of x is less than 1 (one), **acosh()** generates a domain error, returns 0 (zero) and sets global variable [errno](#) to [EDOM](#) ([Mathematical domain error](#)).

See Also[asinh\(\) and _asinh\(\)](#) — Hyperbolic arc sine[atanh\(\) and _atanh\(\)](#) — Hyperbolic arc tangent[cos\(\)](#) — Cosine[coshf\(\)](#) — Hyperbolic cosine using single-precision floating point[coshf\(\)](#) — Hyperbolic cosine using single-precision floating pointGlobal variable [errno](#) — An int used to record error numbers[Transcendental Math Functions](#) on page 424

allocap() and _allocap()

Allocate memory on the stack

Targets

allocap() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_allocap() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

allocap() is Non-ANSI; _allocap() is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <malloc.h>    /* Required */
#include <alloca.h>    /* Required only for alloca */
void * alloca(size_t space);
void * _alloca(size_t space);
```

Description

allocap() and _allocap() allocate *space* bytes of storage in the caller's stack frame. In most implementations of MetaWare C, a call to function allocap() or _allocap() expands into inline code; therefore, the value of *space* is not checked. If *space* has an "irrational" value, the memory stack could be (silently) destroyed.

Return Value

allocap() and _allocap() return a pointer to the allocated block. This storage is automatically freed when the caller returns.

The pointer returned by allocap() and _allocap() will be aligned to an appropriate boundary that suits the target machine (typically, so as to reference a **double**).

System issues

`alloca()` and `_alloca()` are actually implemented as compiler intrinsics. Because `alloca()` is not an ANSI-defined function, the compiler is not normally at liberty to intercept a call to `alloca()`. However, including the file `alloca.h` causes the compiler to recognize `alloca()` as an intrinsic.

See Also

[malloc\(\)](#) — Dynamically allocate uninitialized storage

Example

See the file `alloca.c` in the `install_dir/docs/examples/` directory on your distribution.

asctime() and _asctime()

Convert a **struct tm** to printable form

Targets

`asctime()` and `_asctime()` work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`asctime()` is ANSI compliant; `_asctime()` is Extra-ANSI.

Reentrancy

`asctime()` is Not-Reentrant; `_asctime()` is Reentrant.

Synopsis

```
#include <time.h>          /* Required */
char *asctime(const struct tm *timeptr);
char *_asctime(const struct tm *timeptr, char *buf);
```

Description

`asctime()` converts the time represented by the contents of the **struct** referenced by `timeptr` into a static string of 26 characters. This is the form of the string:

Tue Oct 17 17:04:10 1989

The string is terminated with a newline followed by a NUL.

CAUTION The same static string is overwritten by each call to `asctime()`.

`_asctime()` is equivalent to `asctime()`, except `_asctime()` stores the result in string `buf`.

Return Value

`asctime()` and `_asctime()` return the time in the form of the string described previously.

See Also

[Time-Related Functions](#) on page 434

struct tm — Components of a date and time

Type **time_t** — Representation of a date and time

Example

See the file `asctime.c` in the `install_dir/docs/examples/` directory on your distribution.

asin()

Arc sine

Targets

asin() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */
double asin(double x);
```

Description

asin() computes the inverse sine of x . The domain of **asin()** is $[-1, 1]$. **asin()** might be inlined when the toggle Recognize_library is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

asin() returns the inverse sine of x (assuming x is in the domain $[-1, 1]$). The return value is in the range $[-\pi/2, \pi/2]$.

When x is outside the domain, **asin()** returns 0 (zero) and sets global variable `errno` to [EDOM](#) ([Mathematical domain error](#)).

See Also

[acos\(\)](#) — Arc cosine

[acosf\(\)](#) — Arc cosine using single-precision floating point

[asinf\(\)](#) — Arc sine using single-precision floating point

[asinh\(\) and _asinh\(\)](#) — Hyperbolic arc sine

[atan\(\)](#) — Arc tangent

[sin\(\)](#) — Sine

[sinf\(\)](#) — Sine using single-precision floating point

[sinh\(\)](#) — Hyperbolic sine

[sinhf\(\)](#) — Hyperbolic sine using single-precision floating point

Global variable `errno` — An int used to record error numbers

[Transcendental Math Functions](#) on page 424

Example

See the file `asin.c` in the `install_dir/docs/examples/` directory on your distribution.

asinf()

Arc sine using single-precision floating point

Targets

asinf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */
float asinf(float x);
```

Description

asinf() computes the inverse sine of x . The domain of **asinf()** is $[-1, 1]$. **asinf()** might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

asinf() returns the inverse sine of x (assuming x is in the domain $[-1, 1]$). The return value is in the range $[-\pi/2, \pi/2]$.

When x is outside the domain, **asinf()** returns 0 (zero) and sets global variable **errno** to [EDOM](#) ([Mathematical domain error](#)).

See Also

[acos\(\)](#) — Arc cosine

[acosf\(\)](#) — Arc cosine using single-precision floating point

[sin\(\)](#) — Sine

[sinf\(\)](#) — Sine using single-precision floating point

[asinh\(\)](#) and [asinh\(\)](#) — Hyperbolic arc sine

[atan\(\)](#) — Arc tangent

[atanf\(\)](#) — Arc tangent using single-precision floating point

[sin\(\)](#) — Sine

[sinf\(\)](#) — Sine using single-precision floating point

[sinh\(\)](#) — Hyperbolic sine

[sinhf\(\)](#) — Hyperbolic sine using single-precision floating point

Global variable [errno](#) — [An int used to record error numbers](#)

[Transcendental Math Functions](#) on page 424

asinh() and _asinh()

Hyperbolic arc sine

Targets

asinh() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_asinh() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

asinh() is Non-ANSI; **_asinh()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h> /* Required */
double asinh(double x);
double _asinh(double x);
```

Description

asinh() and **_asinh()** compute the inverse hyperbolic sine of x , using the following calculation:

$$\text{asinh}(x) = \log(x + \sqrt{x^2 + 1})$$

Return Value

asinh() and **_asinh()** return the inverse hyperbolic sine of x .

See Also

[acosh\(\) and _acosh\(\)](#) — [Hyperbolic arc cosine](#)

[atanh\(\) and _atanh\(\)](#) — [Hyperbolic arc tangent](#)

[sin\(\)](#) — [Sine](#)

[sinh\(\)](#) — [Hyperbolic sine](#)

[sinhf\(\)](#) — [Hyperbolic sine using single-precision floating point](#)

[Transcendental Math Functions](#) on page 424

assert()

Abort if an assertion is false

Targets

assert() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <assert.h> /* Required */  
void assert(int expression);
```

Description

If *expression* evaluates to a non-zero value (True), **assert()** does nothing.

If *expression* evaluates to 0 (zero) and the macro **NDEBUG** is not #defined prior to the inclusion of *assert.h*, **assert()** writes the following to standard error and then calls the **abort()** function:

- the name of the file
- the number of the source line where the assertion appears

NOTE **assert()** is provided as a macro only.

See Also

[abort\(\) — Terminate a program abnormally](#)

Example

See the file *assert.c* in the *install_dir/docs/examples/* directory on your distribution.

atan()

Arc tangent

Targets

atan() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */  
double atan(double x);
```

Description

atan() computes the principal value of the arc tangent of *x* in the range $(-\pi/2, \pi/2)$, where *x* is in radians. Because **atan()** is asymptotic about $\pm\pi/2$, its domain is unbounded.

atan() might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

atan() returns the arc tangent of x .

See Also

[acos\(\)](#) — Arc cosine

[asin\(\)](#) — Arc sine

[atan2\(\)](#) — Arc tangent of the angle defined by a point

[atan2f\(\)](#) — Arc tangent of the angle defined by a point, using single-precision floating point

[atanf\(\)](#) — Arc tangent using single-precision floating point

[atanh\(\) and _atanh\(\)](#) — Hyperbolic arc tangent

[tan\(\)](#) — Tangent

[tanf\(\)](#) — Tangent using single-precision floating point

[tanh\(\)](#) — Hyperbolic tangent

[tanhf\(\)](#) — Hyperbolic tangent using single-precision floating point

[Transcendental Math Functions](#) on page 424

Example

See the file `atan.c` in the `install_dir/docs/examples/` directory on your distribution.

atan2()

Arc tangent of the angle defined by a point

Targets

atan2() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

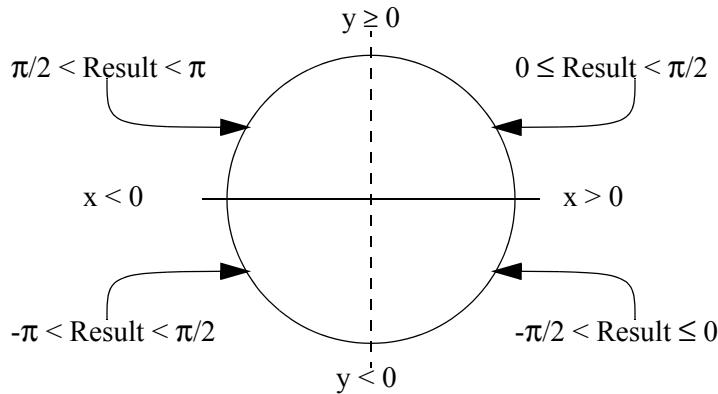
Synopsis

```
#include <math.h>          /* Required */
double atan2(double y, double x);
```

Description

atan2() computes the principal value, in radians, of the arc tangent of y/x in the range $[-\pi, \pi]$. Theoretically, the domain of **atan2()** is unrestricted except that x cannot equal 0 (zero).

The signs of y and x determine the quadrant in which the point (x,y) lies. A line is described by that point and the origin. The result of $\text{atan2}(y,x)$ is the smallest angle from the X axis to that line. Some or all of this angle lies in the first quadrant.



The preceding figure shows how the range of the result can be further defined according to the signs of y and x , as follows:

- If $y \geq 0$ and $x > 0$, result = $[0, \pi/2]$
- If $y \geq 0$ and $x < 0$, result = $(\pi/2, \pi]$
- If $y < 0$ and $x < 0$, result = $(-\pi, -\pi/2)$
- If $y < 0$ and $x > 0$, result = $(-\pi/2, 0]$

atan2() might be inlined when the toggle **Recognize_Library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

atan2 returns the arc tangent of y/x .

If both arguments are 0 (zero), **atan2()** returns 0 (zero) and sets global variable **errno** to [EDOM](#) ([Mathematical domain error](#)).

See Also

[acos\(\)](#) — Arc cosine

[asin\(\)](#) — Arc sine

[atan\(\)](#) — Arc tangent

[atan2f\(\)](#) — Arc tangent of the angle defined by a point, using single-precision floating point

[atanh\(\) and _atanh\(\)](#) — Hyperbolic arc tangent

[tan\(\)](#) — Tangent

[tanf\(\)](#) — Tangent using single-precision floating point

[tanh\(\)](#) — Hyperbolic tangent

[tanhf\(\)](#) — Hyperbolic tangent using single-precision floating point

Global variable **errno** — An int used to record error numbers

[Transcendental Math Functions](#) on page 424

Example

See the file `atan2.c` in the `install_dir/docs/examples/` directory on your distribution.

atan2f()

Arc tangent of the angle defined by a point, using single-precision floating point

Targets

atan2f() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

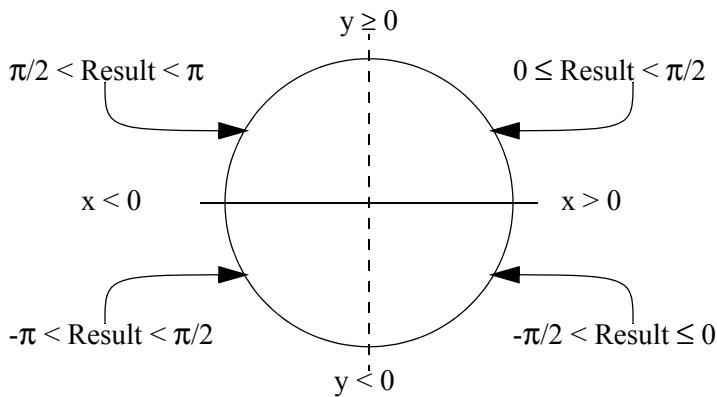
Synopsis

```
#include <math.h>          /* Required */
float atan2f(float y, float x);
```

Description

atan2f() computes the principal value, in radians, of the arc tangent of y/x in the range $[-\pi, \pi]$. Theoretically, the domain of atan2f() is unrestricted except that x cannot equal 0 (zero).

The signs of y and x determine the quadrant in which the point (x,y) lies. A line is described by that point and the origin. The result of atan2f(y,x) is the smallest angle from the X axis to that line. Some or all of this angle lies in the first quadrant.



The preceding figure shows how the range of the result can be further defined according to the signs of y and x , as follows:

- If $y \geq 0$ and $x > 0$, result = $[0, \pi/2]$
- If $y \geq 0$ and $x < 0$, result = $(\pi/2, \pi]$
- If $y < 0$ and $x < 0$, result = $(-\pi, -\pi/2]$
- If $y < 0$ and $x > 0$, result = $(-\pi/2, 0]$

atan2f() might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

atan2f() returns the arc tangent of y/x .

If both arguments are 0 (zero), **atan2f()** returns 0 (zero) and sets global variable **errno** to **EDOM** ([Mathematical domain error](#)).

See Also

[acos\(\)](#) — Arc cosine

[acosf\(\)](#) — Arc cosine using single-precision floating point

[asin\(\)](#) — Arc sine

[asinf\(\)](#) — Arc sine using single-precision floating point

[atan\(\)](#) — Arc tangent

[atan2\(\)](#) — Arc tangent of the angle defined by a point

[atanf\(\)](#) — Arc tangent using single-precision floating point

[atanh\(\) and _atanh\(\)](#) — Hyperbolic arc tangent

[tan\(\)](#) — Tangent

[tanf\(\)](#) — Tangent using single-precision floating point

[tanh\(\)](#) — Hyperbolic tangent

[tanhf\(\)](#) — Hyperbolic tangent using single-precision floating point

Global variable [errno](#) — An int used to record error numbers

[Transcendental Math Functions](#) on page 424

atanf()

Arc tangent using single-precision floating point

Targets

atanf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
float atanf(float x);
```

Description

atanf() computes the principal value of the arc tangent of *x* in the range $(-\pi/2, \pi/2)$, where *x* is in radians. Because **atanf()** is asymptotic about $\pm\pi/2$, its domain is unbounded.

atanf() might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

atanf() returns the arc tangent of x .

See Also

[acos\(\)](#) — Arc cosine

[acosf\(\)](#) — Arc cosine using single-precision floating point

[asin\(\)](#) — Arc sine

[asinf\(\)](#) — Arc sine using single-precision floating point

[atan2\(\)](#) — Arc tangent of the angle defined by a point

[atan2f\(\)](#) — Arc tangent of the angle defined by a point, using single-precision floating point

[atanh\(\) and _atanh\(\)](#) — Hyperbolic arc tangent

[tan\(\)](#) — Tangent

[tanf\(\)](#) — Tangent using single-precision floating point

[tanh\(\)](#) — Hyperbolic tangent

[tanhf\(\)](#) — Hyperbolic tangent using single-precision floating point

[Transcendental Math Functions](#) on page 424

atanh() and _atanh()

Hyperbolic arc tangent

Targets

atanh() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_atanh() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

atanh() is Non-ANSI; **_atanh()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h> /* Required */
double atanh(double x);
double _atanh(double x);
```

Description

atanh() and **_atanh()** compute the inverse hyperbolic tangent of x , using the following calculation:

$$\text{atanh}(x) = 1/2 \log((1+x)/(1-x))$$

Return Value

atanh() and **_atanh()** return the inverse hyperbolic tangent of *x*.

If the value of *x* is not in the range (-1, 1), **atanh()** and **_atanh()** generate a domain error, return 0 (zero), and set global variable **errno** to [EDOM \(Mathematical domain error\)](#).

See Also

[acosh\(\) and _acosh\(\)](#) — [Hyperbolic arc cosine](#)

[asinh\(\) and _asinh\(\)](#) — [Hyperbolic arc sine](#)

[atan\(\)](#) — [Arc tangent](#)

[atan2\(\)](#) — [Arc tangent of the angle defined by a point](#)

[tan\(\)](#) — [Tangent](#)

[tanh\(\)](#) — [Hyperbolic tangent](#)

[tanhf\(\)](#) — [Hyperbolic tangent using single-precision floating point](#)

Global variable [errno](#) — [An int used to record error numbers](#)

[Transcendental Math Functions](#) on page 424

atexit()

Register a function for execution at program termination

Targets

atexit() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */
int atexit(void (*func)(void));
```

Description

atexit() registers a function (pointed to by *func*) that will be called at program termination. The registered function cannot have arguments and does not return a value. Up to 32 functions can be registered with **atexit()**; they are called in reverse order of registration.

Return Value

If the registration succeeds, **atexit()** returns 0 (zero). If the registration fails, **atexit()** returns a non-zero value.

See Also

[exit\(\)](#) — [Terminate a program normally](#)

Example

See the file *atexit.c* in the *install_dir/docs/examples/* directory on your distribution.

atof()

Convert a prefix of a string to floating point

Targets

atof() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */
#include <math.h>
double atof(const char *nptr);
```

Description

atof() converts a prefix of the string *nptr* to a floating-point number. **atof()** recognizes strings of this form:

Wspace? Sign? Dgts ('.' Dgts)? (('e'|'E') Sign? Dgts)?

In this case, Wspace represents Whitespace and Dgts represents Digits (decimal digits), as explained in [Regular Expressions](#) on page 18. The first character that is not in the sequence ends the conversion.

Return Value

atof() returns the prefix of *nptr* converted to a floating-point number.

atof() returns 0 (zero) if any of the following occur:

- The string is empty.
- No Digits are found in the sequence.
- The only Digits found follow the 'e' or 'E'.

The result is undefined if the value cannot be represented.

See Also

[Conversion Functions](#) on page 425

Example

See the file *atof.c* in the *install_dir/docs/examples/* directory on your distribution.

atoi()

Convert a prefix of a string to an **int**

Targets

atol() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */
int atoi(const char *nptr);
```

Description

atol() converts a prefix of the string *nptr* to an integer. **atol()** recognizes strings of this form:

Whitespace? Sign? Digits

The first character that is not in the sequence ends the conversion. (See [Regular Expressions](#) on page 18 for an explanation of this notation.)

Return Value

atol() returns the prefix of *nptr* converted to an integer.

atol() returns 0 (zero) if any of the following occur:

- The string is empty.
- No Digits are found in the sequence.

The result is undefined if the value cannot be represented.

See Also

[**atof\(\)** — Convert a prefix of a string to floating point](#)

[**atol\(\)** — Convert a prefix of a string to a long int](#)

[**atold\(\)** — Convert a prefix of a string to a long double](#)

[**itoa\(\)** and **itoa0\(\)** — Convert an int value to an ASCII string](#)

[Conversion Functions on page 425](#)

Example

See the file *atol.c* in the *install_dir/docs/examples/* directory on your distribution.

atol()

Convert a prefix of a string to a **long int**

Targets

atol() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>          /* Required */  
long int atol(const char *nptr);
```

Description

atol() converts a prefix of the string *nptr* to a long integer. **atol()** recognizes strings of this form:

Whitespace? Sign? Digits

The first character that is not in the sequence ends the conversion. (See [Regular Expressions](#) on page 18 for an explanation of this notation.)

Return Value

atol() returns the prefix of *nptr* converted to an integer.

atol() returns 0 (zero) if any of the following occur:

- The string is empty.
- No Digits are found in the sequence.

The result is undefined if the value cannot be represented.

See Also

[**atof\(\)** — Convert a prefix of a string to floating point](#)

[**atoi\(\)** — Convert a prefix of a string to an int](#)

[**atold\(\)** — Convert a prefix of a string to a long double](#)

[**ltoa\(\) and _ltoa\(\)** — Convert a long int value to an ASCII string](#)

[Conversion Functions](#) on page 425

Example

See the file *atol.c* in the *install_dir/docs/examples/* directory on your distribution.

_atold()

Convert a prefix of a string to a **long double**

Targets

_atold() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>          /* Required */
long double _atold(const char *nptr);
```

Description

`_atold()` converts a prefix of the string `nptr` to a **long double**, and returns that value. `_atold()` recognizes strings of the form:

Whitespace? Sign? Digits

The first character that is not in this sequence ends the conversion (see [Regular Expressions](#) on page 18 for an explanation of this notation).

Return Value

`_atold()` returns the prefix of `nptr` converted to a floating-point number.

`_atold()` returns 0 (zero) if any of the following occur:

- The string is empty.
- No Digits are found in the sequence.

The result is undefined if the value cannot be represented.

See Also

[atof\(\)](#) — Convert a prefix of a string to floating point

[atoi\(\)](#) — Convert a prefix of a string to an int

[atol\(\)](#) — Convert a prefix of a string to a long int

[Conversion Functions](#) on page 425

_bprintf()

Print to a string of given size

Targets

`_bprintf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdio.h>    /* Required */
int _bprintf(char *buf, unsigned int bufsize,
             const char *format, ...);
```

Description

`_bprintf()` is equivalent to `sprintf()`, except that the argument `bufsize` specifies the size of the character array `buf` into which `_bprintf()` places the generated output. `_bprintf()` places a null character at the end of the generated character string. See [printf\(\)](#) for a description of the format string.

Return Value

`_bprintf()` returns the number of characters written into the array, not counting the terminating null character. If an error occurs while converting a value for output, global variable `errno` contains a value indicating the error that has been detected.

If `bufsize` is not large enough to accommodate the characters to be printed, `_bprintf()` sets `errno` to [EIO \(I/O error\)](#).

See Also

[printf\(\) — Print to stdout](#)

Global variable [errno — An int used to record error numbers](#)

[The *printf Family of Functions](#) on page 424

[I/O Functions](#) on page 427

bsearch()

Perform a binary search

Targets

`bsearch()` works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */
void *bsearch(const void *key, const void *base, size_t nmemb, size_t size,
    int (*compare) (const void *ptr1, const void *ptr2));
```

Description

`bsearch()` searches an array for an object that matches a key object.

- *key* points to the key object.
- *base* points to the array of objects being compared to the key object.
- *nmemb* is the number of members in the array.
- *size* is the size of the members of the array.

The contents of the array must be in ascending order according to a user-defined comparison function *compare*. *compare* is called with two pointers (*ptr1* and *ptr2*) that must point to the *key* object and to an array member, in that order.

- If **ptr1* is less than **ptr2*, the *compare* function returns an **int** less than 0 (zero).
- If **ptr1* is greater than **ptr2*, the *compare* function returns an **int** greater than 0 (zero).
- If **ptr1* is equal to **ptr2*, the *compare* function returns 0 (zero).

Return Value

If a match is found, **bsearch()** returns a pointer to the matching member.

If no match is found, **bsearch()** returns [NULL](#).

If two or more members match, which one **bsearch()** returns is indeterminate.

See Also

[lfind\(\) and _lfind\(\)](#) — Search a linear list for a matching value, without modifying the list

[lsearch\(\) and _lsearch\(\)](#) — Search a linear list for a matching value, modifying if necessary

[qsort\(\)](#) — Perform a quicksort

Example

See the file `bsearch.c` in the `install_dir/docs/examples/` directory on your distribution.

cabs() and _cabs()

Compute complex absolute value

Targets

`cabs()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_cabs()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`cabs()` is Non-ANSI; `_cabs()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h> /* Required */
double cabs(struct complex cval);
double _cabs(struct complex cval);
```

Description

`cabs()` and `_cabs()` compute the absolute value of a complex number. The complex number is represented in a structure of type [complex](#). This structure contains a real part *x* and an imaginary part *y*; both values are **double** types.

Return Value

`cabs()` and `_cabs()` return the value of:

```
sqrt((cval.x * cval.x) + (cval.y * cval.y))
```

Normal math errors are detected by `sqrt()` and the arithmetic operations. These errors cause `cabs()` to return [HUGE_VAL](#) and set `errno` to [ERANGE](#) ([Mathematical range error](#)).

See Also[abs\(\)](#) — Calculate the absolute value of an integer[fabs\(\)](#) — Calculate absolute value of a floating-point number using double precision[labs\(\)](#) — Calculate the absolute value of a long int[sqrt\(\)](#) — Square root[complex](#) [complex](#) — A complex numberGlobal variable [errno](#) — An int used to record error numbers**Example**See the file `cabs.c` in the `install_dir/docs/examples/` directory on your distribution.

calloc()

Dynamically allocate zero-initialized storage for objects

Targets

calloc() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

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

Description

calloc() dynamically allocates space in memory for an array of *nelem* elements of *elsize* bytes each. The allocated space is initialized to all zeros.

Return Value

calloc() returns a pointer to the lowest byte of the newly allocated space. If the space cannot be allocated, calloc() returns [NULL](#).

See Also[Memory Allocation and Deallocation Functions](#) on page 434**Example**See the file `calloc.c` in the `install_dir/docs/examples/` directory on your distribution.

ceil()

Calculate smallest integer greater than or equal to a value

Targets

ceil() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */
double ceil(double x);
```

Description

ceil() calculates the smallest integer greater than or equal to x .

Return Value

ceil() returns the value of the ceiling of x .

See Also

[ceilf\(\)](#) — Calculate smallest integer greater than or equal to a value using single-precision floating point

[floor\(\)](#) — Calculate largest integer not greater than a value

[floorf\(\)](#) — Calculate largest integer not greater than a value using single-precision floating point

Example

See the file `ceil.c` in the `install_dir/docs/examples/` directory on your distribution.

ceilf()

Calculate smallest integer greater than or equal to a value using single-precision floating point

Targets

ceilf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */
float ceil(float x);
```

Description

ceilf() calculates the smallest integer greater than or equal to x .

Return Value

`ceil()` returns the value of the ceiling of x .

See Also

[ceil\(\)](#) — Calculate smallest integer greater than or equal to a value

[floor\(\)](#) — Calculate largest integer not greater than a value

[floorf\(\)](#) — Calculate largest integer not greater than a value using single-precision floating point

cgets() and _cgets()

Read a line of text (string) from the keyboard

Targets

`cgets()` and `_cgets()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`cgets()` is Non-ANSI; `_cgets()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <conio.h>
char *cgets(char *strptr);
char *_cgets(char *strptr);
```

Description

`cgets()` and `_cgets()` read one line of text from the keyboard, and place it in the string specified by `strptr`. `strptr` must point to a character array.

- The first element of the array, `strptr[0]`, must contain the maximum number of characters to read into the array.
- The number of characters in the string is placed in the second byte of the array, at `strptr[1]`.
- The characters read are placed in the array, starting at the third byte of the array; that is, at `strptr[2]`. The Enter (Return) key is replaced by the normal zero-byte string terminator.

Return Value

`cgets()` and `_cgets()` return a pointer to the start of the string read from the keyboard, which is always at `strptr[2]`.

See Also

[cputs\(\) and _cputs\(\)](#) — Write a string to the screen

[I/O Functions](#) on page 427

Example

See the file `cgets.c` in the `install_dir/docs/examples/` directory on your distribution.

chdir() and _chdir()

Change the current working directory

Targets

chdir() works on the following target platforms:

EMB	UNIX	DOS	NT
—	x	x	x

_chdir() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

chdir() is Non-ANSI; _chdir() is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
/* Required for DOS and Windows targets: */
#include <direct.h>
/* Required for all other targets: */
#include <unistd.h>

int chdir (char *newpath);
int _chdir (char *newpath);
```

Description

chdir() and _chdir() change the current working directory to the directory specified by *newpath*. These functions do not change the default drive.

For example, assuming you have these current working directories and c: is the current drive:

```
a:\mydir
c:\highc\medium
```

this command makes c:\highc the current directory:

```
chdir ("c:\\highc")
```

but this command changes the working directory on drive A to a:\ without changing the default drive, leaving you in c:\highc\medium:

```
chdir ("a:\\")
```

If your intent is to be left in a:\, you must change the default drive to A before calling chdir().

System issues

On UNIX, this function can be provided by the system library.

Return Value

chdir() and _chdir() return 0 (zero) if the working directory was successfully changed.

If the call results in an error, chdir() and _chdir() return -1 and set global variable errno to [ENOENT](#) ([File or directory not found; pathname element does not exist](#)).

See Also[getcwd\(\) and _getcwd\(\)](#) — Get full pathname of the current working directory[mkdir\(\) and _mkdir\(\)](#) — Create a new directory[rmdir\(\) and _rmdir\(\)](#) — Remove a directoryGlobal variable [errno](#) — An int used to record error numbers**Example**See the file `chdir.c` in the `install_dir/docs/examples/` directory on your distribution.

chmod() and _chmod()

Alter the permission setting for a file

Targets

chmod() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

_chmod() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

chmod() is Non-ANSI; _chmod() is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <sys/types.h>
#include <sys/stat.h>
/* Required for DOS and Windows targets: */
#include <io.h>

/* Required for all other targets: */
#include <unistd.h>

int chmod(char *path, int mode);
int _chmod(char *path, int mode);
```

Descriptionchmod() and _chmod() change the read and write access to the existing file specified in *path* by altering the file's permission setting.*mode*, the new permission setting, must be set to one or both of the following constants defined in `sys/stat.h`: `_S_IWRITE` (write-only permission) or `_S_IREAD` (read-only permission).The combination (`_S_IREAD | _S_IWRITE`) specifies read/write permission. Any other setting is undefined. The permission setting takes effect when the file is closed.

System issues

On DOS and Windows NT, all files are readable (it is not possible to give write-only permission), so the read permission is included for compatibility with other systems, but is ignored here.

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

If successful, `chmod()` and `_chmod()` return 0 (zero). If not successful, these functions return -1 and set global variable `errno` to [ENOENT](#) ([File or directory not found; pathname element does not exist](#)).

See Also

Constants [S_I*](#) — [File permissions](#)

Global variable `errno` — [An int used to record error numbers](#)

[File-Related Functions](#) on page 429

Example

See the file `chmod.c` in the `install_dir/docs/examples/` directory on your distribution.

chsize() and _chsize()

Extend or truncate the length of a file

Targets

`chsize()` and `_chsize()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`chsize()` is Non-ANSI; `_chsize()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <io.h>
int chsize(int handle, long newsize);
int _chsize(int handle, long newsize);
```

Description

`chsize()` and `_chsize()` cause the length of the open file associated with `handle` to be extended or truncated to `newsize`. The file must be opened in a mode that allows writing to the file.

- If extended, the file is padded with NUL characters ('\0').
- If truncated, data after the new end-of-file is lost.

The location of the file pointer is unchanged by `chsize()` and `_chsize()`, even if it is beyond the new end-of-file.

Return Value

`chsize()` and `_chsize()` return 0 (zero) if the size of the file was successfully changed. If not successful, these functions return -1 and set global variable `errno` to one of the following:

[EACCES](#) — [Access error; permission denied](#)

EBADF — “Bad” (invalid or inaccessible) file handle

ENOSPC — No space left on device

An error that sets `errno` to **EACCESS** can mean the file is locked against access, or is read-only (DOS 3.0 and later)

See Also

[File-Related Functions](#) on page 429

Global variable `errno` — An int used to record error numbers

Example

See the file `chsize.c` in the `install_dir/docs/examples/` directory on your distribution.

clearerr()

Clear end-of-file and error flags

Targets

`clearerr()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
void clearerr(FILE *stream);
```

Description

`clearerr()` resets the end-of-file and error flags for the stream associated with `stream`. These flags are cleared when the file is opened, or by explicit calls to `clearerr()` and [rewind\(\)](#). The end-of-file flag is cleared by a call to [fseek\(\)](#).

Note `clearerr()` is provided as a macro and as a function.

See Also

[File-Related Functions](#) on page 429

`typedef struct FILE` — [Information for controlling a stream](#)

clock()

Report elapsed processor time

Targets

clock() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <time.h> /* Required */  
clock_t clock(void);
```

Description

clock() tells how many clock ticks have elapsed since the beginning of an implementation-specific starting point. The return value of **clock()** should be used only for comparison purposes. You can approximate time in seconds by dividing the return value by **CLOCKS_PER_SEC**.

For ARC processors, **clock()** uses the ARC timers if they are available, and so can be used as a cycle counter. Include header file *install_dir/inc/arc/arc_timer.h* to access this functionality. Timer 0 is used by default. To change the default to timer 1, modify the value of **DEFAULT_TIMER** in *install_dir/lib/src/mw/misc_g/arc_timer.c*. Then rebuild the **libmw.a** libraries by executing **gmake mw** from the **lib** directory.

Return Value

clock() returns the product of the elapsed time in seconds and **CLOCKS_PER_SEC**.

See Also

- [Time-Related Functions](#) on page 434
- Type [**clock_t** — Representation of elapsed processor time](#)
- Macro constant [**CLOCKS_PER_SEC** — Clock ticks converted to seconds](#)

Example

See the file **clockspe.c** in the *install_dir/docs/examples/* directory on your distribution.

close() and _close()

Close a file handle

Targets

close() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

_close() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`close()` is Non-ANSI; `_close()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
/* Required for DOS and Windows targets: */
#include <io.h>

/* Required for all other targets: */
#include <unistd.h>

int close(int fhandle);
int _close(int fhandle);
```

Description

`close()` and `_close()` request the operating system to close the file handle specified by `fhandle`. This action also clears the end-of-file flag, and the binary or text mode that might have been associated with this file handle.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

Upon successful completion, `close()` and `_close()` return a value of 0 (zero).

If an error occurs, `close()` and `_close()` return -1 and set global variable `errno` to [EBADF \(“Bad” invalid or inaccessible file handle\)](#).

See Also

[I/O Functions](#) on page 427

[File-Related Functions](#) on page 429

Global variable [errno — An int used to record error numbers](#)

Example

See the file `close.c` in the `install_dir/docs/examples/` directory on your distribution.

_closedir()

Close a POSIX directory stream

Targets

`_closedir()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <dirent.h> /* Required */
int _closedir(_DIR *dir);
```

Description

`_closedir()` closes the directory stream *dir*, which was opened successfully by an earlier call to [_opendir\(\)](#).

Return Value

If the directory stream is successfully closed, the return value is 0 (zero).

If an error occurs and the directory is not closed successfully, `_closedir()` returns -1 and sets global variable `errno` to [EBADF](#) (“Bad” (invalid or inaccessible) file handle).

See Also

[_opendir\(\)](#) — Open a POSIX directory stream

[_readdir\(\)](#) — Read a POSIX directory stream

[_rewinddir\(\)](#) — Reset a POSIX directory stream

Global variable [errno](#) — An int used to record error numbers

complex [_dirent](#) — A POSIX file name for `_readdir()`

Example

See the file `opendir.c` in the `install_dir/docs/examples/` directory on your distribution.

cos()

Cosine

Targets

`cos()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */
double cos(double x);
```

Description

`cos()` calculates the cosine of *x*, where *x* is in radians. The domain is unbounded and the range is [-1, +1]. `cos()` might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer’s Guide* for details.

Return Value

`cos()` returns the cosine of *x* as a **double**.

See Also[acos\(\)](#) — Arc cosine[acosh\(\) and _acosh\(\)](#) — Hyperbolic arc cosine[cosf\(\)](#) — Cosine using single-precision floating point[coshf\(\)](#) — Hyperbolic cosine using single-precision floating point[coshf\(\)](#) — Hyperbolic cosine using single-precision floating point[sin\(\)](#) — Sine[tan\(\)](#) — Tangent[Transcendental Math Functions](#) on page 424**Example**See the file `cos.c` in the `install_dir/docs/examples/` directory on your distribution.

cosf()

Cosine using single-precision floating point

Targets

cosf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */
float cosf(float x);
```

Description

cosf() calculates the cosine of x , where x is in radians. The domain is unbounded and the range is $[-1, +1]$. cosf() might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Valuecosf() returns the cosine of x as a **float**.**See Also**[acos\(\)](#) — Arc cosine[acosf\(\)](#) — Arc cosine using single-precision floating point[acosh\(\) and _acosh\(\)](#) — Hyperbolic arc cosine[cos\(\)](#) — Cosine[coshf\(\)](#) — Hyperbolic cosine using single-precision floating point[coshf\(\)](#) — Hyperbolic cosine using single-precision floating point

[sin\(\)](#) — Sine

[sinf\(\)](#) — Sine using single-precision floating point

[tan\(\)](#) — Tangent

[tanf\(\)](#) — Tangent using single-precision floating point

[Transcendental Math Functions](#) on page 424

cosh()

Hyperbolic cosine

Targets

cosh() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */  
double cosh(double x);
```

Description

cosh() calculates the hyperbolic cosine of x . The domain of this function is unbounded and the range of return values is ≥ 1 (one); that is, $[1, +\infty)$.

cosh() might be inlined when the toggle Recognize_library is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

cosh() returns the hyperbolic cosine of x as a **double**.

If the magnitude of x is too large, such that cosh(x) cannot be represented, cosh() sets global variable errno to [ERANGE](#) ([Mathematical range error](#)) and returns [HUGE_VAL](#).

See Also

[acos\(\)](#) — Arc cosine

[acosf\(\)](#) — Arc cosine using single-precision floating point

[acosh\(\)](#) and [_acosh\(\)](#) — Hyperbolic arc cosine

[cos\(\)](#) — Cosine

[coshf\(\)](#) — Hyperbolic cosine using single-precision floating point

[sinh\(\)](#) — Hyperbolic sine

[tanh\(\)](#) — Hyperbolic tangent

[tanhf\(\)](#) — Hyperbolic tangent using single-precision floating point

Global variable [errno](#) — An int used to record error numbers

[Transcendental Math Functions](#) on page 424

Example

See the file `cosh.c` in the `install_dir/docs/examples/` directory on your distribution.

coshf()

Hyperbolic cosine using single-precision floating point

Targets

`coshf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h> /* Required */
float coshf(float x);
```

Description

`coshf()` calculates the hyperbolic cosine of x . The domain of this function is unbounded and the range of return values is ≥ 1 (one); that is, $[1, +\infty)$.

`coshf()` might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

`coshf()` returns the hyperbolic cosine of x as a `float`.

If the magnitude of x is too large, such that `coshf(x)` cannot be represented, `coshf()` sets global variable `errno` to [ERANGE](#) (Mathematical range error) and returns [HUGE_VAL](#).

See Also

[acos\(\)](#) — Arc cosine

[acosf\(\)](#) — Arc cosine using single-precision floating point

[acosh\(\) and _acosh\(\)](#) — Hyperbolic arc cosine

[cos\(\)](#) — Cosine

[sinh\(\)](#) — Hyperbolic sine

[sinhf\(\)](#) — Hyperbolic sine using single-precision floating point

[tanh\(\)](#) — Hyperbolic tangent

[tanhf\(\)](#) — Hyperbolic tangent using single-precision floating point

Global variable [errno](#) — An int used to record error numbers

[Transcendental Math Functions](#) on page 424

cprintf() and _cprintf()

Print to screen

Targets

cprintf() and **_cprintf()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

cprintf() is Non-ANSI; **_cprintf()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <cconio.h> /* Required */
int cprintf(const char *format, ...);
int _cprintf(const char *format, ...);
```

Description

cprintf() and **_cprintf()** are equivalent to **printf()**, with two exceptions:

- Output is done with **putch()** directly.
- **\n** is not translated to **\r\n** as with **printf()**.

CAUTION Garbage might be printed if the number of arguments passed to **cprintf()** (or **_cprintf()**) is less than the number specified by *format*, or if the types of the arguments do not match the types specified by *format*.

Return Value

On success, **cprintf()** and **_cprintf()** return the number of characters printed. If a write error occurs, **cprintf()** and **_cprintf()** return a negative number.

See Also

[printf\(\) — Print to stdout](#)

[putch\(\) and _putch\(\) — Write a character to the screen](#)

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[I/O Functions](#) on page 427

Example

See the file `cprintf.c` in the `install_dir/docs/examples/` directory on your distribution.

cputs() and _cputs()

Write a string to the screen

Targets

cputs() and **_cputs()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

cputs() is Non-ANSI; **_cputs()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <conio.h> /* Required */
int cputs(char *strptr);
int _cputs(char *strptr);
```

Description

cputs() and **_cputs()** display on the screen the contents of the string specified by *strptr*. Unlike the **puts()** function, **cputs()** and **_cputs()** do not translate the terminating 0 (zero) byte of the string to a carriage return/line feed.

Return Value

Upon successful completion, **cputs()** and **_cputs()** return 0 (zero). If not successful, they return a non-zero value.

See Also

[putch\(\) and _putch\(\)](#) — Write a character to the screen

[puts\(\)](#) — Write a string to standard output

[I/O Functions](#) on page 427

Example

See the file `cputs.c` in the *install_dir/docs/examples/* directory on your distribution.

creat() and _creat()

Create an empty file

Targets

creat() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

_creat() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

creat() is Non-ANSI; **_creat()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <sys/types.h>
#include <sys/stat.h>
/* Required for DOS and Windows targets: */
#include <io.h>
/* Required for all other targets: */
#include <fcntl.h>

int creat(char *pathptr, int permits);
int _creat(char *pathptr, int permits);
```

Description

creat() produces an empty file in the directory with the name specified by *pathptr*.

- If the file does not already exist, it is created, using the permission setting given by *permits*.
- If the file does exist, and its old permission settings allow, its contents are discarded and its length is set to 0 (zero).

The permission-settings field *permits* is used only when a new file is created. *permits* must be set to one or both of the following constants defined in *sys/stat.h*: **_S_IWRITE** (write-only permission) or **_S_IREAD** (read-only permission). The combination (**_S_IREAD** | **_S_IWRITE**) specifies read/write permission. Any other setting is undefined. The permission setting takes effect when the file is closed.

The global file-permission mask is applied to *permits* before it is used. The global mask can be changed with the **umask()** function.

NOTE A call to **open()** with flags **O_CREATE** and **O_TRUNC** (or a call to **_open()** with **_O_CREATE** and **_O_TRUNC**) produces the same result as a call to **creat()** or **_creat()**.

Return Value

Upon successful completion, **creat()** and **_creat()** return a file handle.

If the file cannot be created, **creat()** and **_creat()** return -1 and set global variable **errno** to one of the following values:

EACCES — [Access error; permission denied](#)

EMFILE — [Too many open file handles](#)

ENOENT — [File or directory not found; pathname element does not exist](#)

System issues

On DOS and Windows NT, all files are readable (it is not possible to give write-only permission), so the read permission is included for compatibility with other systems, but is ignored here. **creat()** and **_creat()** always open files in DOS compatibility mode.

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

See Also

[File-Related Functions](#) on page 429

Constants [S_I*](#) — [File permissions](#)

Global variable [errno](#) — [An int used to record error numbers](#)

Example

See the file `creat.c` in the `install_dir/docs/examples/` directory on your distribution.

_crotl() and _crotr()

Rotate a **char** left or right

Targets

The `_crot*()` functions work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Both functions are Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h>
unsigned char _crotl(unsigned char value, int bits);
unsigned char _crotr(unsigned char value, int bits);
```

Description

`_crotl()` rotates *value* to the left by the number of bits given in *bits*.

`_crotr()` rotates *value* to the right by *bits* bits.

Return Value

`_crotl()` returns *value* rotated to the left.

`_crotr()` returns *value* rotated to the right.

See Also

[`lrotl\(\)` and `lrotr\(\)`, `lrotl\(\)` and `lrotr\(\)`](#) — [Rotate a long int left or right](#)

[`rotl\(\)` and `rotr\(\)`, `rotl\(\)` and `rotr\(\)`](#) — [Rotate an int left or right](#)

[`srotl\(\)` and `srotr\(\)`, `srotl\(\)` and `srotr\(\)`](#) — [Rotate a short int left or right](#)

Example

See the file `crot.c` in the `install_dir/docs/examples/` directory on your distribution.

cscanf() and _cscanf()

Read from standard input

Targets

`cscanf()` and `_cscanf()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`cscanf()` is Non-ANSI; `_cscanf()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <cconio.h> /* Required */  
int cscanf(const char *format, ...);  
int _cscanf(const char *format, ...);
```

Description

A call to `cscanf()` or `_cscanf()` is equivalent to calling

```
fscanf(stdin,format,...)
```

except that input is done with `_getche()` directly, not through an input buffer. `cscanf()` and `_cscanf()` echo the input character, unless the last call was to function `_ungetch()`.

CAUTION Garbage might be input if the types of the arguments passed to `cscanf()` (or `_cscanf()`) do not match the types specified by *format*. Garbage might also result if fewer arguments are passed to `cscanf()` or `_cscanf()` than *format* specifies.

Return Value

`cscanf()` and `_cscanf()` return the number of values successfully converted and assigned.

See Also

[scanf\(\) — Read values from `stdin`](#)

[_ungetch\(\) and _ungetch\(\) — Push a character back into the input buffer](#)

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[I/O Functions](#) on page 427

Example

See the file `cscanf.c` in the *install_dir/docs/examples/* directory on your distribution.

ctime() and _ctime()

Convert a `time_t` value to printable form

Targets

`ctime()` and `_ctime()` work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`ctime()` is ANSI compliant; `_ctime()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <time.h> /* Required */
char *ctime(const time_t *timer);
char *_ctime(const time_t *timer, char *buf);
```

Description

`ctime()` converts the time represented by *timer* into a static string of 26 characters. This is the form of the string:

```
Tue Oct 17 17:04:10 1989
```

The string is terminated with a newline followed by a NUL.

`_ctime()` is equivalent to `ctime()`, except `_ctime()` stores the result in string *buf*.

`ctime()` and `_ctime()` use the environment variable TZ to determine the local time zone. (See the description of `tzset()` for an explanation of TZ). If TZ is not set, `ctime()` and `_ctime()` default to Coordinated Universal Time (UTC).

CAUTION The same static string is overwritten by each call to `ctime()` and `asctime()`.

Return Value

`ctime()` and `_ctime()` return the time in the form of a string.

See Also

[Time-Related Functions](#) on page 434

[Environment Variable TZ](#)

Example

See the file `ctime.c` in the *install_dir/docs/examples/* directory on your distribution.

dieeetomsbin() and _dieeetomsbin()

Convert IEEE **double** to Microsoft **double**

Targets

`dieeetomsbin()` and `_dieeetomsbin()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`dieeetomsbin()` is Non-ANSI; `_dieeetomsbin()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h>
int dieeetombsbin(double *ieee_val, double *ms_val);
int _dieeetombsbin(double *ieee_val, double *ms_val);
```

Description

dieeetombsbin() and **_dieeetombsbin()** convert an IEEE-format **double** to a Microsoft-format **double**.

IEEE **double** (eight-byte real):

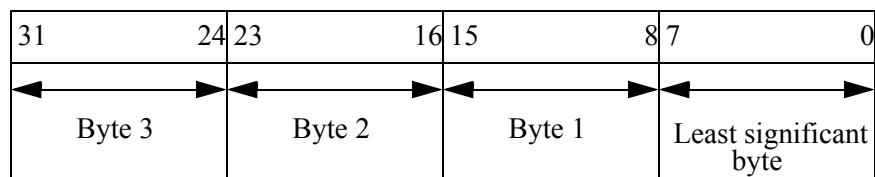
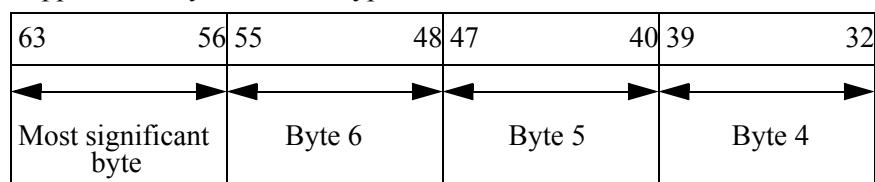
63	62	52	51	0
Sign	11-bit exponent, biased by 1023			52-bit mantissa

Microsoft **double** (eight-byte real):

63	56	55	54	0
8-bit exponent, biased by 129	Sign	55-bit mantissa		

Microsoft floating-point format differs from IEEE floating-point format. Modern versions of Microsoft C compilers operate in IEEE format, but Microsoft Basic and earlier Microsoft C compilers used the Microsoft format to store data that might need to be accessed by a MetaWare C compiler.

Mapped memory for **double** types:



Return Value

dieeetombsbin() and **_dieeetombsbin()** return 0 (zero) when the operation is successful. When the operation is not successful, these functions return 1 (one) to indicate that an overflow occurred.

See Also

[dmsbintoieee\(\)](#) and [dmsbintoieee\(\)](#) — Convert Microsoft double to IEEE double

[fieeetombsbin\(\)](#) and [fieeetombsbin\(\)](#) — Convert IEEE float to Microsoft format

[fmsbintoieee\(\)](#) and [fmsbintoieee\(\)](#) — Convert Microsoft float to IEEE float

Example

See the file *dieeetom.c* in the *install_dir/docs/examples/* directory on your distribution.

difftime()

Calculate difference between two times

Targets

`difftime()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <time.h>          /* Required */
double difftime(time_t time2, time_t time1);
```

Description

`difftime()` calculates the difference between two times.

Return Value

`difftime()` returns the value of `time2 - time1`, expressed in seconds.

See Also

[Time-Related Functions](#) on page 434

Type [`time_t` — Representation of a date and time](#)

Example

See the file `difftime.c` in the `install_dir/docs/examples/` directory on your distribution.

_disable()

Disable interrupts

Targets

`_disable()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <dos.h>
void _disable(void);
```

Description

`_disable()` executes the **CLI** instruction to disable interrupts.

See Also[enable\(\)](#) — Enable interrupts

div()

Perform integer division with remainder

Targets

div() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */
div_t div(int numer, int denom);
```

Description

div() computes the quotient and remainder of the division of *numer* by *denom*.

CAUTION If *denom* is 0 (zero), the program will terminate.

Return Value

div() returns the calculated value in a structure of type **div_t**. If the result cannot be represented, the behavior of div() is undefined.

See Also[ldiv\(\)](#) — Perform long int division with remainder[typedef struct div_t](#) — Result type of function div()**Example**

See the file *div.c* in the *install_dir/docs/examples/* directory on your distribution.

_div0()

Replaceable handler for integer division by zero (ARC only)

Targets

_div0() works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
extern int _div0(int dividend);
```

Description

The run-time library for ARC targets calls `_div0()` when signed integer division detects division by zero. The default version of `_div0()` returns the maximum integer value with the same sign as *dividend*. You can supply your own version to implement the behavior you desire.

Note If you replace `_div0()` with your own functionality, you must also replace `_udiv0()`.

Return Value

`_div0()` returns the maximum integer value with the same sign as the dividend.

See Also

[_udiv0\(\) — Replaceable handler for unsigned integer division by zero \(ARC only\)](#)

dmsbintoieee() and _dmsbintoieee()

Convert Microsoft **double** to IEEE **double**

Targets

`dmsbintoieee()` and `_dmsbintoieee()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`dmsbintoieee()` is Non-ANSI; `_dmsbintoieee()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h>
int dmsbintoieee(double *ms_val, double *ieee_val);
int _dmsbintoieee(double *ms_val, double *ms_val);
```

Description

`dmsbintoieee()` and `_dmsbintoieee()` convert a Microsoft-format **double** to an IEEE-format **double**.

Microsoft floating-point format differs from IEEE floating-point format. Modern versions of Microsoft C compilers operate in IEEE format, but Microsoft Basic and earlier Microsoft C compilers used the Microsoft format to store data that might need to be accessed by a MetaWare C compiler.

See [dieetombsin\(\) and _dieetombsin\(\)](#) for an illustration of mapped memory for **double** types, Microsoft-format **double** types, and IEEE-format **double** types.

Return Value

When the operation is successful, `dmsbintoieee()` and `_dmsbintoieee()` return 0 (zero). When the operation is not successful, these functions return 1 (one) to indicate that an overflow occurred.

See Also

[dieetombsbin\(\) and _dieetombsbin\(\)](#) — Convert IEEE double to Microsoft double

[fieetombsbin\(\) and _fieetombsbin\(\)](#) — Convert IEEE float to Microsoft format

[fmsbintoieee\(\) and _fmsbintoieee\(\)](#) — Convert Microsoft float to IEEE float

Example

See the file `dieetom.c` in the `install_dir/docs/examples/` directory on your distribution.

dup() and _dup()

Duplicate a file handle

Targets

`dup()` and `_dup()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`dup()` is Non-ANSI; `_dup()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <iostream.h>
int dup(int handle);
```

Description

`dup()` and `_dup()` allocate the next available file handle and associate it with the currently open file handle specified by `handle`.

Both handles can then be used to access the file. File positioning action taken with one of the handles is also reflected in the other handle. The type of access allowable on both handles is the same as that of the original file.

Return Value

If successful, `dup()` and `_dup()` return the newly allocated file handle. On failure, both functions return `-1` and set global variable `errno` to one of the following values:

[EBADF](#) — “Bad” (invalid or inaccessible) file handle

[EMFILE](#) — Too many open file handles

See Also

[dup2\(\) and _dup2\(\)](#) — Duplicate one file handle into another

[File-Related Functions](#) on page 429

Example

See the file `dup.c` in the `install_dir/docs/examples/` directory on your distribution.

dup2() and _dup2()

Duplicate one file handle into another

Targets

dup2() and **_dup2()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

dup2() is Non-ANSI; **_dup2()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <io.h>
int dup2(int handle1, int handle2);
```

Description

dup2() and **_dup2()** use a currently open file handle, and associate a second file handle with the same file. Then both handles can be used to access the file. File positioning action taken with one of the handles is also reflected in the other handle. The type of access allowable on both handles is the same as that of the original file.

dup2() and **_dup2()** force *handle2* to refer to the same file as *handle1*. *handle2* is closed if it was open.

Return Value

If successful, **dup2()** and **_dup2()** return 0 (zero). On failure, both functions return -1 and set global variable *errno* to one of the following values:

EBADF — [“Bad” \(invalid or inaccessible\) file handle](#)

EMFILE — [Too many open file handles](#)

See Also

[dup\(\)](#) and [_dup\(\)](#) — [Duplicate a file handle](#)

[File-Related Functions](#) on page 429

Example

See the file *dup.c* in the *install_dir/docs/examples/* directory on your distribution.

ecvt() and _ecvt()

Convert a floating-point number to a character string

Targets

ecvt() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_ecvt() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`ecvt()` is Non-ANSI; `_ecvt()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdlib.h> /* Required */
char *ecvt(double value, int count, int *dpos, int *sign);
char *_ecvt(double value, int count, int *dpos, int *sign);
```

Description

`ecvt()` and `_ecvt()` convert up to *count* digits of the **double** floating-point number *value* into a character string, append a terminating NUL, and return a pointer to the resulting string.

- If the number of digits in *value* is less than *count*, the buffer is padded with zeros.
- If the number of digits in *value* exceeds *count*, the digit string is rounded.

After the call to `ecvt()` or `_ecvt()`, the position of the decimal point is available in *dpos*, and the sign of *value* is available in *sign*.

- *dpos* gives the position of the decimal point relative to the beginning of the string. Positive values indicate the number of digits to the right; negative or zero values indicate that the decimal point is to the left of the first digit.
- If *sign* is 0 (zero), the returned string represents a positive number; otherwise, the number is negative.

Return Value

`ecvt()` and `_ecvt()` return a pointer to a static string of digits. The character string is placed in a static buffer shared by `_ecvt()`, [fcvt\(\) and _fcvt\(\)](#), [_ldecvt\(\)](#), and [_ldfcvt\(\)](#), and is overwritten by the next call to any of these functions.

See Also

[Conversion Functions](#) on page 425

Example

See the file `ecvt.c` in the *install_dir/docs/examples/* directory on your distribution.

_enable()

Enable interrupts

Targets

`_enable()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <dos.h>
void _enable(void);
```

Description

_enable() executes an STI instruction to enable interrupts.

See Also

[_disable\(\)](#) — [Disable interrupts](#)

eof() and _eof()

Test a file handle for end-of-file status

Targets

eof() and _eof() work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

eof() is Non-ANSI; _eof() is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <i0.h>           /* Required */
int _eof(int handle);
```

Description

eof() checks the file-handle table to see that the file is open, then checks to see if the handle is at the end-of-file condition.

Return Value

eof() returns 1 (one) if *handle* is at the end-of-file condition; otherwise, it returns 0 (zero).

If the file associated with *handle* is not open, _eof() returns -1 and sets global variable errno to [EBADF](#) (“Bad” (invalid or inaccessible) file handle).

See Also

[File-Related Functions](#) on page 429

Global variable [errno](#) — [An int used to record error numbers](#)

Example

See the file eof.c in the *install_dir/docs/examples/* directory on your distribution.

exit()

Terminate a program normally

Targets

`_exit()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
void exit(int status);
```

Description

`exit()` causes the program to terminate. Any functions registered with `atexit()` are called in reverse order of registration. Subsequently, open streams are closed and temporary files are removed. *status* is returned to the calling environment of the program.

See Also

[abort\(\)](#) — Terminate a program abnormally

[atexit\(\)](#) — Register a function for execution at program termination

[_exit\(\)](#) — Terminate a program immediately

Example

See the file `exit.c` in the *install_dir/docs/examples/* directory on your distribution.

_exit()

Terminate a program immediately

Targets

`_exit()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>
void _exit(int status);
```

Description

`_exit()` causes the program to terminate immediately. Open streams are not closed, temporary files are not removed, and functions registered with `atexit()` are not called. *status* is returned to the calling environment of the program.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

See Also

- [abort\(\)](#) — Terminate a program abnormally
- [atexit\(\)](#) — Register a function for execution at program termination
- [exit\(\)](#) — Terminate a program normally

exp()

Calculate exponential (e to the power x)

Targets

`exp()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double exp(double x);
```

Description

`exp()` calculates the value of the exponential (e^x). `exp()` might be inlined when the toggle Recognize_library is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

`exp()` returns the value of e^x .

- If x is too large, such that e^x cannot be represented, `exp()` sets `errno` to [ERANGE](#) and returns [HUGE_VAL](#) with the same sign as x .
- If x is too small, such that e^x cannot be represented, `exp()` sets `errno` to [ERANGE](#) and returns 0 (zero).

See Also

- [expf\(\)](#) — Calculate exponential (e to the power x) using single-precision floating point
- [ldexp\(\)](#) — Multiply by a power of 2
- [pow\(\)](#) — Raise a double to a power
- [powf\(\)](#) — Raise a float to a power

Example

See the file `exp.c` in the `install_dir/docs/examples/` directory on your distribution.

expf()

Calculate exponential (e to the power x) using single-precision floating point

Targets

expf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
float expf(float x);
```

Description

expf() calculates the value of the exponential (e^x). **expf()** might be inlined when the toggle Recognize_library is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

expf() returns the value of e^x .

- If x is too large, such that e^x cannot be represented, **expf()** sets `errno` to **ERANGE** and returns **HUGE_VAL** with the same sign as x .
- If x is too small, such that e^x cannot be represented, **expf()** sets `errno` to **ERANGE** and returns 0 (zero).

See Also

[**exp\(\)** — Calculate exponential \(e to the power x\)](#)

[**ldeps\(\)** — Multiply by a power of 2](#)

[**ldepsf\(\)** — Multiply by a power of 2 using single-precision floating point](#)

[**pow\(\)** — Raise a double to a power](#)

[**powf\(\)** — Raise a float to a power](#)

fabs()

Calculate absolute value of a floating-point number using double precision

Targets

fabs() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double fabs(double x);
```

Description

fabs() calculates the absolute value of *x*. **fabs()** might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

NOTE Unless you specify command-line options **-Hansi** or **-Hpcc**, **_abs()** replaces the functions **abs()**, **labs()**, and **fabs()**.

Return Value

fabs() returns the absolute value of *x*.

Surprises

If an **int** is passed to **fabs()**, it is converted to a **double** and **fabs()** computes the absolute value of the converted **int**.

See Also

[**abs\(\)** — Calculate the absolute value of an integer](#)

[**fabsf\(\)** — Calculate absolute value of a floating-point number using single precision](#)

[**labs\(\)** — Calculate the absolute value of a long int](#)

Example

See the file *fabs.c* in the *install_dir/docs/examples/* directory on your distribution.

fabsf()

Calculate absolute value of a floating-point number using single precision

Targets

fabsf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
float fabsf(float x);
```

Description

fabsf() calculates the absolute value of *x*. **fabsf()** might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

fabsf() returns the absolute value of *x*.

Surprises

If an **int** is passed to **fabsf()**, it is converted to a **float** and **fabsf()** computes the absolute value of the converted **int**.

See Also

[abs\(\)](#) — Calculate the absolute value of an integer

[fabs\(\)](#) — Calculate absolute value of a floating-point number using double precision

[labs\(\)](#) — Calculate the absolute value of a long int

fclose()

Close a file

Targets

fclose() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */  
int fclose(FILE *stream);
```

Description

fclose() causes *stream* to be disassociated from the file it has controlled, and closes the file. If the file is open for writing and is buffered, any data in the buffer associated with *stream* is written to the file. If the buffer was automatically allocated, as opposed to being explicitly associated with *stream* by way of [setbuf\(\)](#), it is deallocated.

CAUTION If neither **fclose()** nor [fflush\(\)](#) has been called on a buffered output stream and the program terminates abnormally, the data in the associated buffer at program termination (if any) is not written to the file. An example of abnormal termination is an interrupt from the keyboard; the application would not call **exit()** or return from **main()**.

Return Value

If the operation is successful, **fclose()** returns 0 (zero). If the operation fails, **fclose()** returns **EOF** and sets **errno** to [EBADF](#) ("Bad" (invalid or inaccessible) file handle).

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — [Information for controlling a stream](#)

Constant **EOF** — [End-of-file indicator](#)

Global variable **errno** — [An int used to record error numbers](#)

Example

See the file `fclose.c` in the `install_dir/docs/examples/` directory on your distribution.

fcloseall() and _fcloseall()

Close all files

Targets

`fcloseall()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_fcloseall()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`fcloseall()` is Non-ANSI; `_fcloseall()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>           /* Required */
int fcloseall(void);
int _fcloseall(void);
```

Description

`fcloseall()` and `_fcloseall()` close all open files, except `stdin`, `stdout`, `stderr`. If a file is open for writing and is buffered, any data in the buffer associated with the open file is written to the file. If the buffer was automatically allocated, as opposed to being explicitly associated with the file by way of [`setbuf\(\)`](#), it is deallocated.

Return Value

`fcloseall()` and `_fcloseall()` return the number of files that were closed. If an error is encountered, these functions return `EOF`.

See Also

[File-Related Functions](#) on page 429

Constant **EOF** — [End-of-file indicator](#)

fcvt() and _fcvt()

Convert a floating-point number to a character string

Targets

fcvt() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_fcvt() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

fcvt() is Non-ANSI; **_fcvt()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdlib.h>           /* Required */
char *fcvt(double value, int count, int *dpos, int *sign);
char *_fcvt(double value, int count, int *dpos, int *sign);
```

Description

fcvt() and **_fcvt()** convert a **double** floating-point value to a character string, append a terminating NUL ('\0'), and return a pointer to the resulting string. These functions convert the entire integer portion of *value* and up to *count* digits of the fractional portion.

- If the number of digits in the fractional portion is less than *count*, the buffer is padded with zeros.
- If the number of digits in the fractional portion exceeds *count*, the digit string is rounded.

After the call to **fcvt()** or **_fcvt()**, the position of the decimal point is available in *dpos*, and the sign of *value* is available in *sign*.

- *dpos* gives the position of the decimal point relative to the beginning of the string. Positive values indicate the number of digits to the right; negative or zero values indicate that the decimal point is to the left of the first digit.
- If *sign* is 0 (zero), the returned string represents a positive number; otherwise, the number is negative.

Return Value

fcvt() and **_fcvt()** return a pointer to a static string of digits. The character string is placed in a static buffer shared by [ecvt\(\) and _ecvt\(\)](#), [_fcvt\(\)](#), [_ldecvt\(\)](#), and [_ldfcvt\(\)](#), and is overwritten by the next call to any of these functions.

See Also

[Conversion Functions](#) on page 425

Example

See the file `fcvt.c` in the *install_dir/docs/examples/* directory on your distribution.

fdopen() and _fdopen()

Associate a stream with an open handle

Targets

fdopen() and **_fdopen()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

fdopen() is Non-ANSI; **_fdopen()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <fcntl.h>      /* Required for constants */
#include <stdio.h>
FILE *fdopen(int handle, char *mode);
FILE *_fdopen(int handle, char *mode);
```

Description

fdopen() and **_fdopen()** associate a stream with *handle*, and return a pointer to that stream. *handle* must be a valid open file handle.

The possible values for *mode* are **_TEXT** (open in text mode) and **_BINARY** (open in binary mode). *mode* must match the mode in which *handle* was opened. If *mode* does not match the actual opening mode of *handle*, the behavior that results from using the stream is undefined.

CAUTION The opening *mode* of the stream is not validated against the opening *mode* of the *handle*.

Return Value

fdopen() and **_fdopen()** return a pointer to an open stream. If an error occurs, **fdopen()** and **_fdopen()** return **NULL**.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — Information for controlling a stream

Example

See the file *fdopen.c* in the *install_dir/docs/examples/* directory on your distribution.

feof()

Test for end-of-file

Targets

feof() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
int feof(FILE *stream);
```

Description

feof() tests for the end-of-file on *stream*. Check for a **NULL** pointer before calling **feof()**.

NOTE **feof()** is provided as a macro and as a function.

Return Value

If the end-of-file indicator for the file associated with *stream* is set, **feof()** returns a non-zero value. If the end-of-file indicator is clear, **feof()** returns 0 (zero). If **feof()** is passed a **FILE** pointer that is **NULL**, the behavior is unpredictable.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — [Information for controlling a stream](#)

ferror()

Test for a read or write error on a file

Targets

ferror() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
int ferror(FILE *stream);
```

Description

ferror() tests for read or write errors on a file.

NOTE **ferror()** is provided as a macro and a function.

Return Value

ferror() returns a non-zero value if the error indicator for the file associated with *stream* is set. It returns 0 (zero) if the error indicator is clear.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — [Information for controlling a stream](#)

fflush()

Flush a file's buffer

Targets

fflush() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>          /* Required */
int fflush(FILE *stream);
```

Description

fflush() writes any data in the buffer of the output stream associated with *stream* to the associated file. (The buffer is automatically flushed when full and at normal program termination.)

CAUTION If neither **fflush()** nor **fclose()** has been called on a buffered output stream and the program terminates abnormally, the data in the associated buffer at program termination (if any) is not written to the file. An example of abnormal termination is an interrupt from the keyboard; the application would not call exit or return from main.

Return Value

fflush() returns 0 (zero) when successful. If the file was not open for writing or a write error occurs, **fflush()** returns **NULL** and sets **errno** such that a call to **perror()** will result in an error message.

System issues

The DOS flush operation in the **fflush()** pathway might result in a noticeable slowdown for applications that call **fflush()** often. The explicit flush operation guarantees that all data for a file is flushed to the disk, providing safety in the event of a system crash.

If you do not require periodic flushing to disk, and want better performance, place the following code in your application:

```
int __flush (int fd) {
    return 0;
};
```

To replace **__flush.obj** in library **hc386.lib**, place this **flush()** function in its own compilation unit.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — [Information for controlling a stream](#)

[perror\(\)](#) — [Print an error message](#)

Global variable [errno](#) — [An int used to record error numbers](#)

fgetc()

Read (get) a character from a file

Targets

fgetc() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */  
int fgetc(FILE *stream);
```

Description

fgetc() gets the next character from the input stream *stream*. The associated file pointer (if one is defined) is advanced one character.

Return Value

fgetc() returns the next character from the file associated with *stream*. If the stream is at end-of-file or a read error occurs, **fgetc()** returns **EOF**.

See Also

[typedef struct FILE — Information for controlling a stream](#)

Constant [EOF — End-of-file indicator](#)

[I/O Functions](#) on page 427

Example

See the file `fgetc.c` in the *install_dir/docs/examples/* directory on your distribution.

fgetchar() and _fgetchar()

Get a character from standard input

Targets

fgetchar() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_fgetchar() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

fgetchar() is Non-ANSI; **_fgetchar()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h> /* Required */
int fgetchar(void);
int _fgetchar(void);
```

Description

fgetchar() and **_fgetchar()** are equivalent to **fgetc(stdin)**. The file pointer associated with **stdin** is advanced one character.

Return Value

fgetchar() and **_fgetchar()** return the next character from standard input.

See Also

[fgetc\(\)](#) — Read (get) a character from a file

[I/O Functions](#) on page 427

Example

See the file **fgetchar.c** in the *install_dir/docs/examples/* directory on your distribution.

fgetpos()

Determine file position

Targets

fgetpos() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
int fgetpos(FILE *stream, fpos_t *pos);
```

Description

fgetpos() stores the current file position for *stream* in the object pointed to by *pos*. The value stored is suitable for use in a subsequent call to [fsetpos\(\)](#) to return the file position for *stream* to its value at the time of the call to **fgetpos()**.

Return Value

fgetpos() returns 0 (zero) upon success; otherwise, **fgetpos()** returns a non-zero value and sets **errno** to one of the following values:

[EBADF](#) — “Bad” (invalid or inaccessible) file handle

[EINVAL](#) — Invalid argument given for *stream*

See Also

[File-Related Functions](#) on page 429

typedefstruct FILE — [Information for controlling a stream](#)

Global variable [errno](#) — [An int used to record error numbers](#)

Example

See the file `fgetpos.c` in the `install_dir/docs/examples/` directory on your distribution.

fgets()

Read a line of text (string) from a file

Targets

`fgets()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
char *fgets(char *string, int n, FILE *stream);
```

Description

`fgets()` reads a line of text from the file associated with `stream` into the string `string`, appending a final NUL character ('\0').

- No more than $n-1$ characters are read.
- No characters are read after a newline is encountered. The newline becomes the last (non-NUL) character stored.
- No characters are read after end-of-file is encountered.

Return Value

`fgets()` returns a pointer to `string` if successful. If end-of-file is encountered before any characters have been read, the array is unchanged and `fgets()` returns `NULL`. If a read error occurs, the array contents are indeterminate and `fgets()` returns `NULL`.

[feof\(\)](#) and [ferror\(\)](#) can be used to distinguish between end-of-file and error.

Surprises

`fgets()` and `gets()` are inconsistent: both functions return after encountering a newline, but `fgets()` stores the newline while `gets()` discards it.

See Also

[feof\(\)](#) — [Test for end-of-file](#)

[ferror\(\)](#) — [Test for a read or write error on a file](#)

[gets\(\)](#) — [Read a line of text \(string\) from standard input](#)

typedef **struct FILE** — [Information for controlling a stream](#)

[I/O Functions](#) on page 427

Example

See the file `fgets.c` in the `install_dir/docs/examples/` directory on your distribution.

fieeetombsbin() and _fieeetombsbin()

Convert IEEE **float** to Microsoft format

Targets

`fieeetombsbin()` and `_fieeetombsbin()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`fieeetombsbin()` is Non-ANSI; `_fieeetombsbin()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

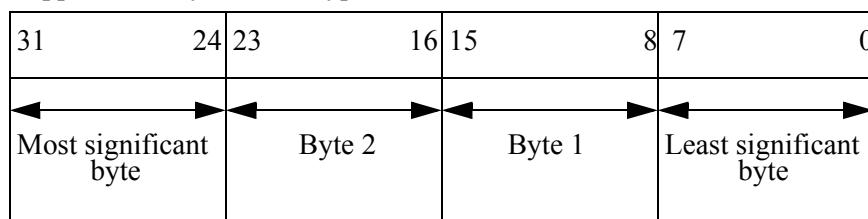
```
#include <math.h>
int fieeetombsbin(float *ieee_val, float *ms_val);
int _fieeetombsbin(float *ieee_val, float *ms_val);
```

Description

`fieeetombsbin()` and `_fieeetombsbin()` convert an IEEE-format **float** value to a Microsoft-format **float** value.

Microsoft floating-point format differs from IEEE floating-point format. Modern versions of Microsoft C compilers operate in IEEE format, but Microsoft Basic and earlier Microsoft C compilers used the Microsoft format to store data that may need to be accessed by a MetaWare C compiler.

Mapped memory for **float** types:



Microsoft **float** (four-byte real):

31	24	23	22	0
8-bit exponent, biased by 129	Sign	23-bit mantissa		

IEEE **float** (four-byte real):

31	30	23	22	0
Sign	8-bit exponent, biased by 127		23-bit mantissa	

Return Value

`fieetombsin()` and `_fieetombsin()` return 0 (zero) on success or 1 (one) if overflow occurs. Both functions set `_matherr()` if an error occurs.

See Also

[`dieeeetombsin\(\)` and `_dieeeetombsin\(\)`](#) — Convert IEEE double to Microsoft double

[`dmsbintoieee\(\)` and `_dmsbintoieee\(\)`](#) — Convert Microsoft double to IEEE double

[`fmsbintoieee\(\)` and `_fmsbintoieee\(\)`](#) — Convert Microsoft float to IEEE float

[`matherr\(\)`, `matherrx\(\)` and `_matherr\(\)`, `_matherrx\(\)`](#) — Provide standard error reporting for math functions

Example

See the file `dieeeetom.c` in the `install_dir/docs/examples/` directory on your distribution.

filelength() and _filelength()

Determine length of an open file

Targets

`filelength()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_filelength()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`filelength()` is Non-ANSI; `_filelength()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>
long filelength(int handle);
long _filelength(int handle);
```

Description

When passed a `handle` associated with an open file, `filelength()` and `_filelength()` determine the length of the file in bytes.

Return Value

If successful, **filelength()** and **_filelength()** return the length of the file. If not successful, these functions return -1L and set **errno** to **EBADF** (“Bad” (invalid or inaccessible) file handle).

See Also

[File-Related Functions](#) on page 429

Example

See the file `fileleng.c` in the `install_dir/docs/examples/` directory on your distribution.

fileno() and _fileno()

Get file handle currently associated with a stream

Targets

fileno() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	X	X

_fileno() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

fileno() is Non-ANSI; **_fileno()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdio.h>
int fileno(FILE *stream);
int _fileno(FILE *stream);
```

Description

fileno() and **_fileno()** return the file handle currently associated with *stream*. If more than one handle is associated with the specified stream, the return value is the handle assigned when the stream was opened.

Return Value

fileno() and **_fileno()** return the file handle associated with *stream*. Behavior is undefined if *stream* does not refer to an open file.

If an error occurs, **fileno()** and **_fileno()** return -1.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — Information for controlling a stream

Example

See the file `fileno.c` in the `install_dir/docs/examples/` directory on your distribution.

_findclose() and _findfirst() and _findnext()

Find a file whose name matches a specification

Targets

The _find*() functions work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The _find*() functions are Extra-ANSI.

Reentrancy

The _find*() functions are Not-Reentrant.

Synopsis

```
#include <io.h> /* Required */
long _findfirst(char *filespec, struct finddata_t *fileinfo);
int _findnext(long handle, struct finddata_t *fileinfo);
int _findclose(long handle);
```

Description

The _find*() functions find files whose names match a specification you provide.

_findfirst() finds the first file whose name matches *filespec*. You can use the '*' and '?' wildcard characters in *filespec*. If it finds the specified file, _findfirst() returns information about the file in the [finddata_t](#) structure pointed to by *fileinfo*.

_findnext() finds the next occurrence, if any, of a file specified by the *filespec* argument in a prior call to _findfirst(). Argument *fileinfo* must be initialized by a prior call to _findfirst().

_findclose() closes the internal directory handle allocated by a previous call to _findfirst(). Use the _findclose() function when you no longer need a [finddata_t](#) structure.

NOTE You cannot limit _findfirst() and _findnext() by specifying the attributes of the target file.

The attributes of the target file are returned in the *attrib* field of the [finddata_t](#) structure.

Possible values of *attrib* are [A_ARCH](#), [A_HIDDEN](#), [A_NORMAL](#), [A_RDONLY](#), [A_SYSTEM](#), [A_SUBDIR](#), and [A_VALID](#). For more information about these A_* attribute constants, see [Constants and Globals](#) on page 319.

Return Value

On success:

- _findfirst() returns a search handle that identifies the file or group of files matching the argument *filespec*. Use this handle when making calls to functions _findnext() and _findclose().
- _findnext() and _findclose() return 0 (zero).

On failure, _findfirst(), _findnext(), and _findclose() return -1 and set the global variable *errno* as follows:

[EINVAL](#) — Invalid argument given

ENOENT — (`_findfirst()` only) [File or directory not found; pathname element does not exist](#)

See Also

[File-Related Functions](#) on page 429

struct `finddata_t` — [Attributes of a file](#)

Constants [`A_*`](#) — [File attributes](#)

Global variable [`errno`](#) — [An int used to record error numbers](#)

Example

See the file `find.c` in the `install_dir/docs/examples/` directory on your distribution.

floor()

Calculate largest integer not greater than a value

Targets

`floor()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double floor(double x);
```

Description

`floor()` calculates the largest integer not greater than x .

Return Value

`floor()` returns the largest integer not greater than x .

See Also

[`ceil\(\)`](#) — [Calculate smallest integer greater than or equal to a value](#)

[`ceilf\(\)`](#) — [Calculate smallest integer greater than or equal to a value using single-precision floating point](#)

[`floorf\(\)`](#) — [Calculate largest integer not greater than a value using single-precision floating point](#)

floorf()

Calculate largest integer not greater than a value using single-precision floating point

Targets

floorf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>          /* Required */  
float floorf(float x);
```

Description

floorf() calculates the largest integer not greater than *x*.

Return Value

floorf() returns the largest integer not greater than *x*.

See Also

[**ceil\(\)** — Calculate smallest integer greater than or equal to a value](#)

[**ceilf\(\)** — Calculate smallest integer greater than or equal to a value using single-precision floating point](#)

[**floor\(\)** — Calculate largest integer not greater than a value](#)

flushall() and _flushall()

Clear input buffers and flush output buffers

Targets

flushall() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_flushall() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

flushall() is Non-ANSI; **_flushall()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>    /* Required */  
int flushall();  
int _flushall();
```

Description

flushall() and **_flushall()** clear all buffers belonging to input files, and write any buffers belonging to output files.

Return Value

flushall() and **_flushall()** return the number of open files.

See Also

[File-Related Functions](#) on page 429

fmod()

Return floating-point remainder

Targets

fmod() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double fmod(double x, double y);
```

Description

fmod() calculates the floating-point remainder r of x/y . r has the same sign as x such that $x = (i * y) + r$ for some integer i , where $|r| < |y|$.

Return Value

fmod() returns the floating-point remainder of x/y . If x/y cannot be represented, the result is undefined.

See Also

[fmodf\(\)](#) — [Return floating-point remainder using single precision](#)

[frexp\(\)](#) — [Break a double into fraction and exponent](#)

[modf\(\)](#) — [Break a double into integer and fractional parts](#)

fmodf()

Return floating-point remainder using single precision

Targets

fmodf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
float fmodf(float x, float y);
```

Description

fmodf() calculates the floating-point remainder r of x/y . r has the same sign as x such that $x = (i * y) + r$ for some integer i , where $|r| < |y|$.

Return Value

fmodf() returns the floating-point remainder of x/y . If x/y cannot be represented, the result is undefined.

See Also

[fmod\(\)](#) — Return floating-point remainder

[frexp\(\)](#) — Break a double into fraction and exponent

[frexpf\(\)](#) — Break a float into fraction and exponent

[modf\(\)](#) — Break a double into integer and fractional parts

[modff\(\)](#) — Break a float into integer and fractional parts

fmsbintoieee() and _fmsbintoieee()

Convert Microsoft **float** to IEEE **float**

Targets

fmsbintoieee() and **_fmsbintoieee()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

fmsbintoieee() is Non-ANSI; **_fmsbintoieee()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h>
int fmsbintoieee(float *ms_val, float *ieee_val);
int _fmsbintoieee(float *ms_val, float *ieee_val);
```

Description

fmsbintoieee() and **_fmsbintoieee()** convert Microsoft-format **float** types to IEEE-format **float** types.

Microsoft floating-point format differs from IEEE floating-point format. Modern versions of Microsoft C compilers operate in IEEE format, but Microsoft Basic and earlier Microsoft C compilers used the Microsoft format to store data that may need to be accessed by a MetaWare C compiler.

See [_fieetombsbin\(\) and _fieetombsbin\(\)](#) for an illustration of mapped memory for **float** types, Microsoft-format **float** types, and IEEE-format **float** types.

Return Value

fmsbintoieee() and **_fmsbintoieee()** return 0 (zero) on success, or 1 (one) if underflow has occurred. Both functions set **_matherr()** on error.

See Also

[_dieetombsbin\(\) and _dieetombsbin\(\)](#) — Convert IEEE double to Microsoft double

[dmsbintoieee\(\) and _dmsbintoieee\(\)](#) — Convert Microsoft double to IEEE double

[fieetombsbin\(\) and _fieetombsbin\(\)](#) — Convert IEEE float to Microsoft format

[_matherr\(\), _matherrx\(\) and _matherr\(\), _matherrx\(\)](#) — Provide standard error reporting for math functions

Example

See the file `dieetom.c` in the *install_dir/docs/examples/* directory on your distribution.

fopen()

Open a file

Targets

fopen() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
FILE *fopen(const char *name, const char *mode);
```

Description

fopen() opens the file named by the string *name*. The string *mode* indicates how the file is to be opened.

File Opening Mode

mode must contain one (but only one) of **r**, **w**, or **a**. *mode* can also contain **+**.

r — Open existing file for reading.

r+ — Open existing file for update.

w — Create file for writing; if file exists, truncate it.

w+ — Create file for update; if file exists, truncate it.

a — Open or create file for appending to end of file.

a+ — Open or create file for update, appending to end of file.

mode can also contain one, but only one, of **b** (binary), **t** (text), or **u** (unbuffered).

CAUTION If *mode* contains **w** and *name* contains an existing file name, that file is destroyed.

Unbuffered Versus Buffered Streams

If the *mode* string contains **u**, or if the file being opened references the keyboard or screen, the resulting stream is unbuffered. That is, each read or write request results in a system call to perform I/O.

If the *mode* string does not contain **u**, and the file being opened does not reference the keyboard or the screen, upon the first read or write request, an automatically allocated buffer is associated with the stream and I/O is done in blocks. Prior to the first read or write request, **setbuf()** or **setvbuf()** can be called to associate a particular buffer with the stream and avoid any automatic allocation.

Binary Versus Text Streams

If the *mode* string contains **b**, the resulting stream is binary; if the string contains **t** instead, the result is a text stream. See the **System issues** section for the distinction between text and binary streams. If the string contains both **t** and **b**, the last one in the string takes precedence.

If the *mode* string does not contain **t** or **b**, the default is text on UNIX. On DOS and Windows NT, the default depends on the variable **_fmode**. (The possible values for **_fmode** are discussed in the description of [setmode\(\)](#)).

Read, Write, Append, and Update

If a non-existent file is opened with *mode* string **r**, the open fails.

If an existing file is opened with *mode* string **w**, its contents are destroyed.

If a file is opened with *mode* string **a**, all writing is done at the end of the file, irrespective of the buffer and the position of the file pointer at the time of the write request. The file pointer can be positioned using **fseek()** or **rewind()** for read requests from a file with *mode* string **a+**.

If a file is open for update (*mode* contains **+**), both reading and writing are allowed. When switching between reading and writing on a stream, a call to **fseek()** or **rewind()** is required on buffered files to clear or flush the buffer.

Return Value

fopen() returns a pointer to a **FILE** variable associated with the file. This file variable controls the newly created input or output stream associated with the file. If the file cannot be opened, **fopen()** returns **NULL** and sets global variable **errno** to indicate that an error occurred.

System issues

On DOS and Windows NT, if the stream is a text stream, DOS line terminators (**\r\n**) are converted into C terminators (**\n**). In addition, a CTRL-Z (**\032**) is interpreted as an end-of-file indicator and **EOF** is returned. If the stream is a binary stream, no such conversion occurs, and CTRL-Z characters encountered in the input stream are not interpreted but returned as is.

Due to these conversions, input from an unbuffered text stream might be slow. Where speed is important, we recommend buffering input text streams.

If the file being opened references the keyboard or the screen, the stream is unbuffered; that is, each read or write request results in a system call to perform I/O.

On UNIX, **fopen()** treats text files and binary files identically.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — Information for controlling a stream

Example

See the file `fgets.c` in the `install_dir/docs/examples/` directory on your distribution.

FP_OFF(), FP_SEG() and _FP_OFF(), _FP_SEG()

Get offset and segment of a pointer

Targets

The **FP_*** and **_FP_*** macros are defined on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The **FP_*** macros are Non-ANSI; the **_FP_*** macros are Extra-ANSI.

Reentrancy

The **FP_*** and **_FP_*** macros are Not-Reentrant.

Synopsis

```
#include <dos.h>           /* Required */
unsigned  FP_OFF (void _Far *address);
unsigned  FP_SEG (void _Far *address);
unsigned _FP_OFF (void _Far *address);
unsigned _FP_SEG (void _Far *address);
```

Description

The **FP_*** and **_FP_*** macros return the offset and segment, respectively, of the far pointer *address*.

Return Value

FP_OFF and **_FP_OFF** return the offset of *address*.

FP_SEG and **_FP_SEG** return the segment of *address*.

Example

See the file `fpoff.c` in the `install_dir/docs/examples/` directory on your distribution.

fprintf()

Print to a file

Targets

fprintf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
int fprintf(FILE *stream, const char *format, ...);
```

Description

fprintf() writes to the file associated with *stream* according to the format string *format*. A call to **fprintf(stdout,format,...)** is equivalent to **printf(format)**. For a complete description of the format string, see [printf\(\)](#).

CAUTION Garbage might be printed if the number of arguments passed to **fprintf()** is less than the number specified by *format*, or if the types of the arguments to **fprintf()** do not match the types specified by *format*.

Return Value

fprintf() returns the number of characters printed, or a negative number if a write error occurs.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[I/O Functions](#) on page 427 ([File I/O](#))

typedef struct FILE — Information for controlling a stream

Example

See the file **fprintf.c** in the *install_dir/docs/examples/* directory on your distribution.

fputc()

Write a character to a file

Targets

fputc() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
int fputc(int c, FILE *stream);
```

Description

fputc() writes the character *c* (the least significant byte of the **int** *c*) to the file associated with *stream* at the current position of the file pointer (if one is defined) and advances the file pointer (if any).

Return Value

fputc() returns the character *c* unless a write error occurs, in which case it returns **EOF**.

Surprises

fputc() takes *c* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to **fputc()**, it is automatically coerced to type **int**.

See Also

[I/O Functions](#) on page 427

typedef struct FILE — Information for controlling a stream

Constant [EOF — End-of-file indicator](#)

Example

See the file `fgetc.c` in the *install_dir/docs/examples/* directory on your distribution.

fputchar() and _fputchar()

Write a character to standard output

Targets

fputchar() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	X	X

_fputchar() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

fputchar() is Non-ANSI; **_fputchar()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>
int fputchar(int c);
int _fputchar(int c);
```

Description

fputchar(*c*) and **_fputchar(*c*)** are equivalent to **fputc(*c*, stdout)**.

Return Value

fputchar() and **_fputchar()** return the character that is written to **stdout**. If an error occurs, **fputchar()** and **_fputchar()** return **EOF**.

Surprises

fputchar() and **_fputchar()** take *c* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to **fputchar()**, the object is automatically coerced to type **int**.

See Also

[fputc\(\)](#) — Write a character to a file

[I/O Functions](#) on page 427

Constant [EOF](#) — End-of-file indicator

Example

See the file `fgetchar.c` in the *install_dir/docs/examples/* directory on your distribution.

fputs()

Write a string to a file

Targets

fputs() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>          /* Required */
int fputs(const char *s, FILE *stream);
```

Description

fputs() writes the string *s* to the file associated with *stream* at the current position of the file pointer (if one is defined) and advances the file pointer (if any). The terminating NUL is not written.

Return Value

fputs() returns 0 (zero) if the operation is successful. If **fputs()** fails, it returns **EOF** and sets **errno** such that a call to **perror()** results in an error message describing the reason for the failure.

Surprises

fputs() and **puts()** are inconsistent: **fputs()** writes exactly the string *s*, while **puts()** writes *s* followed by a newline.

See Also

[I/O Functions](#) on page 427

[perror\(\)](#) — Print an error message

[puts\(\)](#) — Write a string to standard output

typedef struct FILE — Information for controlling a stream

Constant [EOF](#) — End-of-file indicator

Example

See the file `fgets.c` in the *install_dir/docs/examples/* directory on your distribution.

fread()

Read from a file

Targets

fread() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
size_t fread(void *ptr, size_t size, size_t nelem, FILE *stream);
```

Description

fread() reads a specified number of elements from an input stream and places them in a specified memory location.

ptr points to the beginning of the segment of memory into which the elements will be placed.

size is the size, in bytes, of the elements.

nelem is the number of elements to be read.

stream points to the input stream from which the elements will be read.

The file pointer, if one is defined, is advanced by the number of bytes read. If a partial element is read (due either to a read error or reaching end-of-file), the element's value is indeterminate.

The first read or write request on a buffered stream causes a buffer to be allocated for the stream unless **setbuf()** has previously been called to associate a buffer with the stream.

Note **fread()** assigns *nelem* * *size* bytes — or as many bytes as it was able to read — into the specified segment of memory.

Return Value

fread() returns the number of elements read, not counting any partial element.

If the return value of **fread()** is less than *nelem*, either a read error has occurred or end-of-file has been encountered.

- If a read error has occurred, the position of the file pointer is indeterminate and the error flag is set for the stream.
- If end-of-file has been encountered and no bytes were read, the end-of-file flag is set for the stream.

feof() and/or **ferror()** must be used to distinguish an error condition from an end-of-file condition.

See Also

[I/O Functions](#) on page 427

[File-Related Functions](#) on page 429

typedef struct FILE — Information for controlling a stream

Example

See the file `fread.c` in the `install_dir/docs/examples/` directory on your distribution.

free()

Release storage allocated by [calloc\(\)](#), [malloc\(\)](#), or [realloc\(\)](#)

Targets

`free()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */  
void free(void *ptr);
```

Description

`free()` takes a pointer to the lowest byte of dynamically allocated storage (a pointer that was returned by [calloc\(\)](#), [malloc\(\)](#), or [realloc\(\)](#)). The storage is made available again for allocation by one of these functions.

CAUTION If the freed storage is referenced, or if the argument to `free()` is not a pointer previously allocated by [calloc\(\)](#), [malloc\(\)](#), or [realloc\(\)](#), the resulting behavior is undefined, and quite likely undesirable.

See Also

[Memory Allocation and Deallocation Functions](#) on page 434

freopen()

Open a file using an existing FILE variable

Targets

`freopen()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>      /* Required */
FILE *freopen(const char *pathname, const char *type, FILE *stream);
```

Description

freopen() closes the file associated with *stream*, then opens the file named by the string *pathname* and associates *stream* with it. The *type* argument is used as in **fopen()**. Failure to close the file originally associated with *stream* is ignored.

The rationale behind **freopen()** is that a new file can be associated with a stream without informing all processes that have copies of the stream value. For example, **freopen()** provides a way to disassociate **stdout** from the console and associate it with a chosen file.

Return Value

If **freopen()** is successful, it returns *stream*. If not successful, it returns **NULL**.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — Information for controlling a stream

Example

See the file *freopen.c* in the *install_dir/docs/examples/* directory on your distribution.

frexp()

Break a **double** into fraction and exponent

Targets

frexp() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>      /* Required */
double frexp(double value, int *exp);
```

Description

frexp() calculates a **double** *value* with magnitude in the range [0.5, 1.0) and stores an **int** in the object referenced by *exp* such that

$$\text{value} = x \times (2.0^{\text{exp}})$$

Return Value

frexp() returns a **double** *value* with the described magnitude.

If *value* is 0 (zero), both the return value and the value stored in **exp* are set to 0 (zero).

See Also

[frexpf\(\)](#) — Break a float into fraction and exponent

[modf\(\)](#) — Break a double into integer and fractional parts

[modff\(\)](#) — Break a float into integer and fractional parts

frexpf()

Break a **float** into fraction and exponent

Targets

frexpf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
float frexpf(float value, int *exp);
```

Description

frexpf() calculates a **float** *value* with magnitude in the range [0.5, 1.0) and stores an **int** in the object referenced by *exp* such that

$$value = x * (2.0^{\text{exp}})$$

Return Value

frexpf() returns a **float** *value* with the described magnitude.

If *value* is 0 (zero), both the return value and the value stored in **exp* are set to 0 (zero).

See Also

[frexpf\(\)](#) — Break a float into fraction and exponent

[modf\(\)](#) — Break a double into integer and fractional parts

[modff\(\)](#) — Break a float into integer and fractional parts

fscanf()

Read values from a file

Targets

fscanf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>          /* Required */
int fscanf(FILE *stream, const char *format, ...);
```

Description

fscanf() reads from the file associated with *stream* according to the format string *format*, assigning values from the file into objects pointed to by subsequent arguments to **fscanf()**. For a detailed description of the format string, see [scanf\(\)](#).

CAUTION If an argument is not the expected pointer, it is still treated as if it were, potentially causing some arbitrary place in memory to be overwritten with the result of the conversion.

If fewer arguments are passed to **fscanf()** than *format* specifies, arbitrary places in memory are overwritten with the results of the specified conversions.

System issuesSee the [System issues](#) section in the entry for [scanf\(\)](#).**Return Value**

fscanf() returns the number of input values assigned, or **EOF** if end-of-file is encountered before the first conflict or conversion. If no conflict occurs, **fscanf()** returns when the end of the format string is encountered.

See Also[The *printf Family of Functions](#) on page 424[The *scanf Family of Functions](#) on page 424[I/O Functions](#) on page 427 ([File I/O](#))[typedef struct FILE — Information for controlling a stream](#)**Example**See the file `fscanf.c` in the *install_dir/docs/examples/* directory on your distribution.

fseek()

Move file pointer to new location in file

Targets**fseek()** works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>          /* Required */
int fseek(FILE *stream, long offset, int ptrname);
```

Description

`fseek()` moves the file pointer for the file associated with *stream* to *offset* bytes from one of three positions specified by the value of *ptrname*; *SEEK_SET* (beginning of file), *SEEK_CUR* (current position), and *SEEK_END* (end of file).

The results of a seek past the end of a file, or a negative seek past the beginning of a file, are system-dependent. A successful seek clears the end-of-file flag and undoes any effect of [ungetc\(\)](#) for the stream. *offset* can be positive or negative.

Return Value

`fseek()` returns 0 (zero) unless the request is invalid. In most systems, a seek outside the limits of a file is invalid and `fseek()` returns -1.

See Also

[File-Related Functions](#) on page 429

[SEEK_CUR and SEEK_END and SEEK_SET](#), [SEEK_CUR and SEEK_END and SEEK_SET](#),
[SEEK_CUR and SEEK_END and SEEK_SET](#) — Macros used to specify position in file

`typedef struct FILE` — [Information for controlling a stream](#)

Example

See the file `fseek.c` in the *install_dir/docs/examples/* directory on your distribution.

fsetpos()

Set file position

Targets

`fsetpos()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>    /* Required */
int fsetpos(FILE *stream, const fpos_t *pos);
```

Description

`fsetpos()` sets the file position of *stream* according to the value stored in the object pointed to by *pos*, which must have been obtained by a previous call to [fgetpos\(\)](#).

If successful, the end-of-file indicator is cleared and any effect of [ungetc\(\)](#) is nullified for *stream*.

Return Value

If successful, **fsetpos()** returns 0 (zero). If not successful, it sets global variable **errno** to a non-zero value and returns a non-zero value.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — Information for controlling a stream

Example

See the file *fgetpos.c* in the *install_dir/docs/examples/* directory on your distribution.

ftell()

Determine current value of a file pointer

Targets

ftell() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>          /* Required */
long ftell(FILE *stream);
```

Description

ftell() calculates the current position of a file pointer. The value is the distance of the pointer's position from the beginning of the file, expressed in terms of characters.

Return Value

ftell() returns the value of the file pointer for the file associated with *stream*. If an error occurs, **ftell()** returns -1.

See Also

[fseek\(\)](#) — [Move file pointer to new location in file](#)

[rewind\(\)](#) — [Seek to the beginning of a file](#)

typedef struct FILE — Information for controlling a stream

Example

See the file *fseek.c* in the *install_dir/docs/examples/* directory on your distribution.

ftime() and _ftime()

Get current time values

_fullpath()

Targets

`ftime()` and `_ftime()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`ftime()` is Non-ANSI; `_ftime()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <sys\types.h>          /* Required */  
#include <sys\timeb.h>  
void ftime(struct timeb *ptr);  
void _ftime(struct timeb *ptr);
```

Description

`ftime()` and `_ftime()` call upon the [time\(\)](#) function to obtain the current time, and store it in the **timeb** structure pointed to by *ptr*. `ftime()` and `_ftime()` also fill in the structure's *timezone* and *dstflag* fields from their global variable equivalents.

Return Value

`ftime()` and `_ftime()` do not return any value. The **timeb** structure passed as an argument is filled in, and all results are passed back in this structure.

See Also

[Time-Related Functions](#) on page 434

struct timeb — [Time-zone information from function ftime\(\)](#)

Example

See the file `ftime.c` in the *install_dir/docs/examples/* directory on your distribution.

_fullpath()

Convert relative pathname to absolute pathname

Targets

`_fullpath()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>    /* Required */  
char * _fullpath(char *buffer, const char *relpath, int buflen);
```

Description

`_fullpath()` converts a relative pathname, such as `..\\stdio.h`, to its absolute pathname, such as `c:\\highc\\inc\\stdio.h`.

- *buffer* points to an array of characters that stores the absolute pathname. If *buffer* is `NULL`, `_fullpath()` allocates a character string of maximum length `MAX_PATH`, defined in `stdlib.h`.
- The character string pointed to by *buffer* can store up to *buflen* characters.
- *relpath* points to an array of characters that contains the relative pathname.

Return Value

If the operation is successfully completed, `_fullpath()` returns a pointer to *buffer*, containing the absolute path.

When *relpath* contains an invalid drive letter, or when *buffer* is not big enough to store the absolute pathname, `_fullpath()` returns `NULL`.

See Also

[_makepath\(\)](#) — Create a path string from components

[_splitpath\(\)](#) — Split a pathname into its components

Constant `MAX_*` — Maximum length of several pathname components

Example

See the file `fullpath.c` in the `install_dir/docs/examples/` directory on your distribution.

fwrite()

Write to a file

Targets

`fwrite()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
size_t fwrite(const void *ptr, size_t size, size_t nelem, FILE *stream);
```

Description

`fwrite()` writes a specified number of elements from a specified memory location to a specified output stream.

- *ptr* points to the beginning of the segment of memory from which the elements will be written.
- *size* is the size, in bytes, of the elements.
- *nelem* is the number of elements to be written.
- *stream* points to the output stream to which the elements will be written.

If a write error occurs, **fwrite()** returns having written fewer than *nelem* elements. The file pointer (if one is defined) is advanced by the number of bytes written. If a write error occurs, the position of the file pointer (if any) is indeterminate.

The first read or write request on a buffered stream causes a buffer to be allocated for the stream, unless [**setbuf\(\)**](#) has been previously called to associate a buffer with the stream.

NOTE **fwrite()** writes *nelem* * *size* bytes — or as many bytes as it was able to — from the specified segment of memory.

Return Value

fwrite() returns the number of elements actually written. If a write error occurs, **fwrite()** returns a value less than *nelem*.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — [Information for controlling a stream](#)

Example

See the file `fread.c` in the *install_dir/docs/examples/* directory on your distribution.

gcvt() and _gcvt()

Convert a floating-point number to a character string

Targets

gcvt() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_gcvt() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

gcvt() is Non-ANSI; **_gcvt()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdlib.h> /* Required */
char *gcvt(double value, int sigdig, char *buffer);
char *_gcvt(double value, int sigdig, char *buffer);
```

Description

gcvt() converts the floating-point number *value* into a character string and stores the resulting string in *buffer*. *buffer* should be large enough to hold the resulting string and terminating NUL.

gcvt() attempts to produce *sigdig* significant digits. If unsuccessful, the function produces *sigdig* significant digits in exponential format.

Return Value

`gcvt()` returns a pointer to the string of converted digits.

See Also

[Conversion Functions](#) on page 425

Example

See the file `gcvt.c` in the `install_dir/docs/examples/` directory on your distribution.

getc()

Get a character from a file

Targets

`getc()` works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
int getc(FILE *stream);
```

Description

`getc()` is equivalent to [fgetc\(\)](#). When `getc()` is used as a macro, `stream` is evaluated more than once.

Note `getc()` is provided both as a macro and as a function.

Return Value

`getc()` returns the next character from the input stream `stream`. The associated file pointer (if one is defined) is advanced one character. If the stream is at end-of-file or a read error occurs, `getc()` returns `EOF`.

See Also

[I/O Functions](#) on page 427

Constant [EOF — End-of-file indicator](#)

`typedef struct FILE — Information for controlling a stream`

Example

See the file `getc.c` in the `install_dir/docs/examples/` directory on your distribution.

getchar()

Get a character from standard input

Targets

getchar() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>
int getchar(void);
```

Description

getchar() is equivalent to [fgetc\(stdin\)](#); see [fgetc\(\)](#).

NOTE **getchar()** is provided both as a macro and as a function. The macro **getchar()** expands to **getc(stdin)**, which itself might be a macro.

Return Value

getchar() returns the next character from the input stream **stdin**. If a read error occurs, **getchar()** returns **EOF**.

See Also

[I/O Functions](#) on page 427

[getc\(\) — Get a character from a file](#)

Constant [EOF — End-of-file indicator](#)

Example

See the file `getchar.c` in the `install_dir/docs/examples/` directory on your distribution.

getcwd() and _getcwd()

Get full pathname of the current working directory

Targets

getcwd() works on the following target platforms:

EMB	UNIX	DOS	NT
—	x	x	x

_getcwd() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

getcwd() is Non-ANSI; _getcwd() is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <direct.h> /* Required */
char *getcwd(char *cwd, int maxlen);
char *_getcwd(char *cwd, int maxlen);
```

Description

getcwd() and **_getcwd()** store the full pathname of the current working directory into *cwd*. The argument *maxlen* specifies the longest pathname *cwd* can hold. If the length of the pathname (including the terminating null character) is larger than *maxlen*, an error occurs.

If *cwd* is **NULL**, **getcwd()** and **_getcwd()** allocate enough memory to hold the current working directory. Because this memory is allocated through a call to **malloc()**, this memory can be freed later by calling **free()** with the return value of **getcwd()** or **_getcwd()** as an argument.

System issues

On UNIX, this function can be provided by the system library.

Return Value

If successful, **getcwd()** and **_getcwd()** return *cwd*, a pointer to the buffer holding the pathname of the current working directory.

If an error occurs, **getcwd()** and **_getcwd()** return **NULL** and set the global variable **errno** to one of the following values:

ENOMEM — [Not enough memory, or invalid DOS arena headers](#)

ERANGE — [Mathematical range error](#) (the length of the pathname is greater than *maxlen*)

See Also

[chdir\(\) and _chdir\(\)](#) — [Change the current working directory](#)

[mkdir\(\) and _mkdir\(\)](#) — [Create a new directory](#)

[rmdir\(\) and _rmdir\(\)](#) — [Remove a directory](#)

Example

See the file `getcwd.c` in the *install_dir/docs/examples/* directory on your distribution.

getenv()

Get the value of an environment variable

Targets

getenv() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>          /* Required */
char *getenv(const char *name);
```

Description

getenv() searches the list of environment variables for the entry corresponding to *name*.

Each program inherits from its parent process a data structure called an *environment block*. The environment block is a series of null-terminated strings of the form *name=parameter* that define the environment in which a process executes.

For more information about the environment block, see function [putenv\(\) and _putenv\(\)](#).

Return Value

getenv() returns a pointer to a string associated with the environment-block entry for *name*. This string is an internal buffer that cannot be modified by the program, but can be overwritten by a subsequent call to **getenv()**.

If *name* is not currently defined in the environment block, **getenv()** returns **NULL**.

See Also

[putenv\(\) and _putenv\(\)](#) — Add or modify an environment variable

Example

See the file *getenv.c* in the *install_dir/docs/examples/* directory on your distribution.

getpid() and _getpid()

Get unique ID of the calling process

Targets

getpid() and **_getpid()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

getpid() is Non-ANSI; **_getpid()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <process.h>
int getpid(void);
int _getpid(void);
```

Description

getpid() and **_getpid()** determine the unique process ID of the calling process. The process ID is an integer that is not duplicated by any other active process ID.

Return Value

getpid() and **_getpid()** return the process ID of the calling process. There is no error return.

Example

See the file *getpid.c* in the *install_dir/docs/examples/* directory on your distribution.

gets()

Read a line of text (string) from standard input

Targets

gets() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
char *gets(char *string);
```

Description

gets() reads a line of text from the stream `stdin` into the string `string`, appending a final NUL. No characters are read after end-of-file or after a newline is encountered. The newline, if any, is discarded.

NOTE Make sure `string` is large enough to contain all input up to and including a newline. If `string` is not large enough, memory beyond `string` might be overwritten.

Return Value

If successful, **gets()** returns `string`. If end-of-file is encountered before any characters have been read, the array is unchanged and **gets()** returns `NULL`. If a read error occurs, the array contents are indeterminate and **gets()** returns `NULL`.

Surprises

gets() and **fgets()** are inconsistent: both functions return after encountering a newline, but **fgets()** stores the newline while **gets()** discards it.

See Also

[fgets\(\)](#) — Read a line of text (string) from a file

[I/O Functions](#) on page 427

Example

See the file `gets.c` in the `install_dir/docs/examples/` directory on your distribution.

getw() and _getw()

Get an integer from a file

Targets

getw() and **_getw()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`getw()` is Non-ANSI; `_getw()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h> /* Required */
int getw(FILE *stream);
int _getw(FILE *stream);
```

Description

`getw()` and `_getw()` get the next integer from a file associated with *stream* and increment the associated file pointer to point to the next unread character.

Return Value

`getw()` and `_getw()` return the next integer from the file.

See Also

[getc\(\)](#) — Get a character from a file

[getchar\(\)](#) — Get a character from standard input

[putw\(\)](#) and [_putw\(\)](#) — Write an integer to a file

[File-Related Functions](#) on page 429

[typedef struct FILE](#) — Information for controlling a stream

Example

See the file `putw.c` in the *install_dir/docs/examples/* directory on your distribution.

gmtime() and _gmtime()

Convert a `time_t` value to a `struct tm`, adjusted to UTC

Targets

`gmtime()` and `_gmtime()` work on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

`gmtime()` is ANSI compliant; `_gmtime()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <time.h> /* Required */
struct tm *gmtime(const time_t *timer);
struct tm *_gmtime(const time_t *timer, struct tm *tbuf);
```

Description

`gmtime()` converts the time represented by *timer* into a `struct` containing the individual components of the time and date, expressed as Coordinated Universal Time (UTC). A pointer to the `struct` is returned.

_gmtime() is equivalent to gmtime(), except _gmtime() stores the result in **tm** structure *tbuf*, as opposed to a static structure.

CAUTION The structure *tbuf* is overwritten by each call to _gmtime().

Return Value

gmtime() and _gmtime() return a pointer to a struct **tm**.

System issues

On UNIX, gmtime() and _gmtime() depend on environment variable TZ. If TZ is not set, the function defaults to Coordinated Universal Time.

See Also

[Time-Related Functions](#) on page 434

Type [**time_t**](#) — [Representation of a date and time](#)

struct [**tm**](#) — [Components of a date and time](#)

[Environment Variable TZ](#)

Example

See the file `gmtime.c` in the *install_dir/docs/examples/* directory on your distribution.

_heapchk()

Check heap for consistency

Targets

_heapchk() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <malloc.h>           /* Required */  
int _heapchk(void);
```

Description

_heapchk() walks through the heap and checks the attributes of each block, such as pointers, size, and flags.

Return Value

_heapchk() returns one of the following constant values: [**HEAPBADBEGIN**](#), [**HEAPBADNODE**](#), [**HEAPEMPTY**](#), or [**HEAPOK**](#). (See the [HEAP*](#) constants in [Constants and Globals](#) on page 319.)

See Also

[heapset\(\)](#) — [Fill free memory locations in the heap](#)

[heapwalk\(\)](#) — [Walk through the heap](#)

Example

See the file `heapwalk.c` in the `install_dir/docs/examples/` directory on your distribution.

_heapset()

Fill free memory locations in the heap

Targets

`_heapset()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <malloc.h> /* Required */  
int _heapset(unsigned int fill);
```

Description

`_heapset()` checks the heap for consistency, then fills free memory locations with the value of `fill`. By setting each byte of each free entry in the heap to `fill`, you can tell which memory locations of the heap contain free nodes and which locations contain unintentionally written data.

Return Value

`_heapset()` returns one of the following constant values: [HEAPBADBEGIN](#), [HEAPBADNODE](#), [HEAPEMPTY](#), or [HEAPOK](#). (See the [HEAP*](#) constants in [Constants and Globals](#) on page 319.)

See Also

[heapchk\(\)](#) — [Check heap for consistency](#)

[heapwalk\(\)](#) — [Walk through the heap](#)

Example

See the file `heapwalk.c` in the `install_dir/docs/examples/` directory on your distribution.

_heapwalk()

Walk through the heap

Targets

`_heapwalk()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <malloc.h>          /* Required */
int _heapwalk(_HEAPINFO *next_entry);
```

Description

`_heapwalk()` walks through the heap one entry at a time.

To call `_heapwalk()`, pass a pointer to a structure of type `_HEAPINFO`. To make the first call to `_heapwalk()`, set the field `_pentry` to `NULL`.

Return Value

`_heapwalk()` returns one of the following constant values: `HEAPBADBEGIN`, `HEAPBADNODE`, `HEAPEMPTY`, `HEAPEND`, or `HEAPOK`. (See the `HEAP*` constants in [Constants and Globals](#) on page 319.)

See Also

[`heapchk\(\)`](#) — Check heap for consistency

[`heapset\(\)`](#) — Fill free memory locations in the heap

`struct _HEAPINFO` — Information about the heap

Example

See the file `heapwalk.c` in the `install_dir/docs/examples/` directory on your distribution.

hypot() and _hypot()

Hypotenuse of a triangle

Targets

`hypot()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_hypot()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`hypot()` is Non-ANSI; `_hypot()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h>          /* Required */
double hypot(double side1, double side2);
```

Description

`hypot()` computes the length of the hypotenuse of a right triangle, using as input the lengths of the two sides that form the right angle.

Return Value

hypot() returns the length of the hypotenuse. Normal math error checking is in effect, and will detect overflow if appropriate. If an error occurs, **hypot()** sets **errno** to [ERANGE \(Mathematical range error\)](#) and returns [HUGE_VAL](#).

See Also

[cos\(\)](#) — [Cosine](#)

[sin\(\)](#) — [Sine](#)

[tan\(\)](#) — [Tangent](#)

Global variable [errno](#) — [An int used to record error numbers](#)

Example

See the file `hypot.c` in the `install_dir/docs/examples/` directory on your distribution.

_initcopy()

Copy initial values from ROM to RAM

Targets

`_initcopy()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
int _initcopy(void);
```

Description

`_initcopy()` supports the ELF linker's **INITDATA** command by copying the initial data values and literal sections from ROM to RAM. See the *ELF Linker and Utilities User's Guide* for more information about the **INITDATA** command.

CAUTION To use `_initcopy()`, the target's ROM must be readable as data.

_inline()

Place bytes of code in line

Targets

`_inline()` works on the following target platforms, on 32-bit Intel 386-family processors:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis`_inline(...);`**Description**

Compiler intrinsic `_inline(e_1, e_2, \dots, e_n)` places the least significant byte of constant expressions e_1, e_2, \dots, e_n directly in the instruction stream. This can be used, for example, to insert a **CLI** or **STI** instruction directly in an interrupt routine.

System issues

This intrinsic function is available only for the 32-bit Intel 386-family processors. Keep this in mind if portability across different platforms is an issue.

Example

See the file `inline.c` in the `install_dir/docs/examples/` directory on your distribution.

_invalidate_dcache()

Invalidate data cache (ARC only)

Targets

`_invalidate_dcache()` works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
extern void _invalidate_dcache(void);
```

Description

Use this function to invalidate the data cache on your ARC processor.

Note You might need to modify this function to operate correctly with your ARC processor configuration. See the file `misc_g/invcache.s`.

Compare the toggle `invalidate_dcache` in the *MetaWare Debugger User's Guide*.

See Also

[_invalidate_icache\(\)](#) — [Invalidate instruction cache \(ARC only\)](#)

_invalidate_icache()

Invalidate instruction cache (ARC only)

Targets

_invalidate_icache() works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
extern void _invalidate_icache(void);
```

Description

Use this function to invalidate the instruction cache on your ARC processor.

NOTE You might need to modify this function to operate correctly with your ARC processor configuration. See the file `misc_g/invcache.s`.

See Also

[_invalidate_dcache\(\)](#) — [Invalidate data cache \(ARC only\)](#)

isalnum()

Test for alphanumeric character

Targets

isalnum() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <cctype.h> /* Required for macro */
int isalnum(int c);
```

Description

isalnum() tests the integer value *c* for an alphanumeric character.

NOTE `isalnum()` is provided both as a macro and as a function.

Return Value

`isalnum()` returns a non-zero value if *c* is a letter or digit. If *c* is not a letter or digit, `isalnum()` returns 0 (zero).

Example

See the file `isalnum.c` in the *install_dir/docs/examples/* directory on your distribution.

isalpha()

Test for alphabetic character

Targets

`isalpha()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h> /* Required for macro */
int isalpha(int c);
```

Description

`isalpha()` tests the integer value *c* for an alphabetic character.

NOTE `isalpha()` is provided both as a macro and as a function.

Return Value

`isalpha()` returns a non-zero value if *c* is a letter. If *c* is not a letter, `isalpha()` returns 0 (zero).

Example

See the file `isalpha.c` in the *install_dir/docs/examples/* directory on your distribution.

isascii() and _isascii()

Test for ASCII character

Targets

`isascii()` and `_isascii()` work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`isascii()` is Non-ANSI; `_isascii()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <ctype.h>
int isascii(int c);
int _isascii(int c);
```

Description

`isascii()` and `_isascii()` test the integer value *c* for an ASCII character.

NOTE `_isascii()` is provided as a function and as a macro.

`isascii()` is provided as a macro.

Return Value

`isascii()` and `_isascii()` return a non-zero value if *c* is an ASCII character. If *c* is not an ASCII character, `isascii()` and `_isascii()` return 0 (zero).

Example

See the file `isascii.c` in the *install_dir/docs/examples/* directory on your distribution.

isatty() and _isatty()

Identify character devices

Targets

`isatty()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

`_isatty()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`isatty()` is Non-ANSI; `_isatty()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
/* Required for DOS and Windows targets: */
#include <iо.h>
/* Required for all other targets: */
#include <unistd.h>
```

```
int isatty (int handle);
int _isatty (int handle);
```

Description

isatty() and **_isatty()** determine whether *handle* refers to a character device such as a terminal, console, printer, or serial port.

isatty() and **_isatty()** can be used to determine if any of the predefined file handles **stdin**, **stdout**, **stderr**, **stdaux**, and **stdprn** are redirected to files.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

isatty() and **_isatty()** return a non-zero value if *handle* is a character device. If *handle* is not a character device, **isatty()** and **_isatty()** return 0 (zero).

Example

See the file **isatty.c** in the *install_dir/docs/examples/* directory on your distribution.

iscntrl()

Test for control character

Targets

iscntrl() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>           /* Required for macro */
int iscntrl(int c);
```

Description

iscntrl() tests the integer value *c* for a control character. A control character is any character in the character set that is not a letter, digit, punctuator, or the space character.

Note **iscntrl()** is provided both as a macro and as a function.

Return Value

iscntrl() returns a non-zero value if *c* is a control character. If *c* is not a control character, **iscntrl()** returns 0 (zero).

Example

See the file **iscntrl.c** in the *install_dir/docs/examples/* directory on your distribution.

isdigit()

Test for numeric character

Targets

isdigit() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>           /* Required for macro */
int isdigit(int c);
```

Description

isdigit() tests the integer value *c* for a numeric character.

NOTE **isdigit()** is provided both as a macro and as a function.

Return Value

isdigit() returns a non-zero value if *c* is a digit. If *c* is not a digit, **isdigit()** returns 0 (zero).

Example

See the file *isdigit.c* in the *install_dir/docs/examples/* directory on your distribution.

isgraph()

Test for visible character

Targets

isgraph() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>           /* Required for macro */
int isgraph(int c);
```

Description

isgraph() tests the integer value *c* for a visible character.

NOTE `isgraph()` is provided both as a macro and as a function.

Return Value

`isgraph()` returns a non-zero value if c is a printable character other than space. If c is a space or a non-printable character, `isgraph()` returns 0 (zero).

Example

See the file `isgraph.c` in the `install_dir/docs/examples/` directory on your distribution.

islower()

Test for lowercase alphabetic character

Targets

`islower()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>           /* Required for macro */
int islower(int c);
```

Description

`islower()` tests the integer value c for a lowercase alphabetic character.

NOTE `islower()` is provided both as a macro and as a function.

Return Value

`islower()` returns a non-zero value if c is a lowercase letter. If c is not a lowercase letter, `islower()` returns 0 (zero).

Example

See the file `islower.c` in the `install_dir/docs/examples/` directory on your distribution.

_isodigit()

Test for octal numeric character

Targets

`_isodigit` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>
int _isodigit(int c);
```

Description

_isodigit tests the integer value *c* for an octal numeric character.

NOTE _isodigit is provided only as a macro.

Return Value

_isodigit returns a non-zero value if *c* is an octal digit. If *c* is not an octal digit, _isodigit returns 0 (zero).

Example

See the file *isodigit.c* in the *install_dir/docs/examples/* directory on your distribution.

isprint()

Test for printable character

Targets

isprint() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h> /* Required for macro */
int isprint(int c);
```

Description

isprint() tests the integer value *c* for a printable character.

NOTE **isprint()** is provided both as a macro and as a function.

Return Value

isprint() returns a non-zero value if *c* is a printable character (including space). If *c* is not a printable character (nor a space), **isprint()** returns 0 (zero).

Example

See the file *isprint.c* in the *install_dir/docs/examples/* directory on your distribution.

ispunct()

Test for punctuation character

Targets

ispunct() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>          /* Required for macro */
int ispunct(int c);
```

Description

ispunct() tests the integer value *c* for a punctuation character.

Note **ispunct()** is provided both as a macro and as a function.

Return Value

ispunct() returns a non-zero value if *c* is a printing character that is not a digit, letter, or space. If *c* is a digit, letter, or space, or a non-printing character, **ispunct()** returns 0 (zero).

Example

See the file *ispunct.c* in the *install_dir/docs/examples/* directory on your distribution.

isspace()

Test for whitespace character

Targets

isspace() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>          /* Required for macro */
int isspace(int c);
```

Description

`isspace()` tests the integer value c for a whitespace character. A whitespace character is defined as one of the following:

- space (' ')
- form feed ('\f')
- horizontal tab ('\t')
- newline ('\n')
- carriage return ('\r')
- vertical tab ('\v')

NOTE `isspace()` is provided both as a macro and as a function.

Return Value

`isspace()` returns a non-zero value if c is a whitespace character. If c is not a whitespace character, `isspace()` returns 0 (zero).

Example

See the file `isspace.c` in the `install_dir/docs/examples/` directory on your distribution.

isupper()

Test for uppercase alphabetic character

Targets

`isupper()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h> /* Required for macro */
int isupper(int c);
```

Description

`isupper()` tests the integer value c for an uppercase alphabetic character.

NOTE `isupper()` is provided both as a macro and as a function.

Return Value

`isupper()` returns a non-zero value if c is an uppercase letter. If c is not an uppercase letter, `isupper()` returns 0 (zero).

Example

See the file `isupper.c` in the `install_dir/docs/examples/` directory on your distribution.

isxdigit()

Test for hexadecimal numeric character

Targets

`isxdigit()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h> /* Required for macro */
int isxdigit(int c);
```

Description

`isxdigit()` tests the integer value *c* for a hexadecimal numeric character.

Note `isxdigit()` is provided both as a macro and as a function.

Return Value

`isxdigit()` returns a non-zero value if *c* is a hexadecimal digit. If *c* is not a hexadecimal digit, `isxdigit()` returns 0 (zero).

Example

See the file `isxdigit.c` in the `install_dir/docs/examples/` directory on your distribution.

itoa() and _itoa()

Convert an **int** value to an ASCII string

Targets

`itoa()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_itoa()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`itoa()` is Non-ANSI; `_itoa()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h>          /* Required */
char *itoa(int value, char *target, int radix);
char *_itoa(int value, char *target, int radix);
```

Description

`itoa()` converts *value* into the ASCII string *target*. The conversion is performed according to the number system specified by *radix*.

radix can be any number between 2 and 36. If *radix* is the commonly used decimal 10, a negative *value* will have a leading minus sign. On a machine where `int` is 32 bits and *radix* is 2, the maximum size for *target* is 33 bytes, including the terminating null byte.

Return Value

`itoa()` returns a pointer to the start of the result string.

See Also

[Conversion Functions](#) on page 425

Example

See the file `itoa.c` in the *install_dir/docs/examples/* directory on your distribution.

j0(), j1(), jn() and _j0(), _j1(), _jn()

Bessel function of the first kind

Targets

The `j*`() functions work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

The `_j*`() functions work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The `j*`() functions are Non-ANSI; the `_j*`() functions are Extra-ANSI.

Reentrancy

The `j*`() and `_j*`() functions are Reentrant.

Synopsis

```
#include <math.h>          /* Required */
double j0(double x);
double j1(double x);
double jn(int pow, double x);
```

```
double _j0(double x);
double _j1(double x);
double _jn(int pow, double x);
```

Description

j0() and **_j0()** calculate a Bessel function of the first kind, power 0.

j1() and **_j1()** calculate a Bessel function of the first kind, power 1.

jn() and **_jn()** calculate a Bessel function of the first kind, integral power *pow*.

If *pow* is 0, **jn()** (or **_jn()**) calls **j0()** (or **_j0()**). If *pow* is 1, **jn()** (or **_jn()**) calls **j1()** (or **_j1()**).

Return Value

Return result of **j0()** or **_j0()** is a Bessel function of *x* to power 0.

Return result of **j1()** or **_j1()** is a Bessel function of *x* to power 1.

Return result of **jn()** or **_jn()** is a Bessel function of *x* to integral power *pow*.

See Also

[**y0\(\), y1\(\), yn\(\) and _y0\(\), _y1\(\), _yn\(\)**](#) — [Bessel function of the second kind](#)

Example

See the file **j.c** in the *install_dir/docs/examples/* directory on your distribution.

labs()

Calculate the absolute value of a **long int**

Targets

labs() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */
long labs(long i);
```

Description

labs() computes the absolute value of *i*.

Note Unless you specify command-line options **-Hansi** or **-Hpcc**, **abs()** replaces the functions **abs()**, **labs()**, and **fabs()**.

Return Value

labs() returns the absolute value of *i*.

Surprises

labs(LONG_MIN) returns **LONG_MIN** because there is no corresponding positive **long int**.

See Also

[abs\(\)](#) — Calculate the absolute value of an integer

[cabs\(\) and _cabs\(\)](#) — Compute complex absolute value

[fabs\(\)](#) — Calculate absolute value of a floating-point number using double precision

Example

See the file `labs.c` in the `install_dir/docs/examples/` directory on your distribution.

[_ldecvt\(\)](#)

Convert a **long double** to a character string

Targets

[_ldecvt\(\)](#) works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h> /* Required */  
char *_ldecvt(long double value, int count, int *dpos, int *sign);
```

Description

[_ldecvt\(\)](#) converts up to *count* digits of the **long double** floating-point number *value* into a character string, appends a terminating NUL character ('\0'), and returns a pointer to the resulting string.

- If the number of digits in *value* is less than *count*, the buffer is padded with zeros.
- If the number of digits in *value* exceeds *count*, the digit string is rounded.

After the call to [_ldecvt\(\)](#), the position of the decimal point is available in *dpos*, and the sign of *value* is available in *sign*.

- *dpos* gives the position of the decimal point relative to the beginning of the string. Positive values indicate the number of digits to the right; negative or zero values indicate that the decimal point is to the left of the first digit.
- If *sign* is 0 (zero), the returned string represents a positive number. Otherwise, the number is negative.

Return Value

[_ldecvt\(\)](#) returns a pointer to a static string of digits. The character string is placed in a static buffer shared by [ecvt\(\)](#) and [_ecvt\(\)](#), [fcvt\(\)](#) and [_fcvt\(\)](#), [_ldecvt\(\)](#), and [_ldfcvt\(\)](#), and is overwritten by the next call to any of these functions.

See Also

[Conversion Functions](#) on page 425

Idexp()

Multiply by a power of 2

Targets

Idexp() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double ldexp(double x, int exp);
```

Description

Idexp() computes $x * 2^{exp}$.

Return Value

Idexp() returns $x * 2^{exp}$. If the result exceeds **HUGE_VAL** in magnitude, **Idexp()** sets **errno** to **ERANGE** and the return value has magnitude **HUGE_VAL** with the same sign as x .

See Also

[exp\(\)](#) — Calculate exponential (e to the power x)

[expf\(\)](#) — Calculate exponential (e to the power x) using single-precision floating point

[ldexpf\(\)](#) — Multiply by a power of 2 using single-precision floating point

[pow\(\)](#) — Raise a double to a power

[powf\(\)](#) — Raise a float to a power

Example

See the file `ldexp.c` in the *install_dir/docs/examples/* directory on your distribution.

Idexpf()

Multiply by a power of 2 using single-precision floating point

Targets

Idexpf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>          /* Required */
float ldexpf(float x, int exp);
```

Description

ldexpf() computes $x * 2^{exp}$.

Return Value

ldexpf() returns $x * 2^{exp}$. If the result exceeds **HUGE_VAL** in magnitude, **ldexpf()** sets errno to **ERANGE** and the return value has magnitude **HUGE_VAL** with the same sign as x .

See Also

[exp\(\)](#) — Calculate exponential (e to the power x)

[expf\(\)](#) — Calculate exponential (e to the power x) using single-precision floating point

[ldexp\(\)](#) — Multiply by a power of 2

[pow\(\)](#) — Raise a double to a power

[powf\(\)](#) — Raise a float to a power

_ldfcvt()

Convert a **long double** to a character string

Targets

_ldfcvt() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>    /* Required */
char *_ldfcvt(long double value, int count, int *dpos, int *sign);
```

Description

_ldfcvt() converts a **long double** floating-point value to a character string, appends a terminating NUL character ('\0'), and returns a pointer to the resulting string.

_ldfcvt() converts the entire integer portion of *value* and up to *count* digits of the fractional portion.

- If the number of digits in the fractional portion is less than *count*, the buffer is padded with zeros.
- If the number of digits in the fractional portion exceeds *count*, the digit string is rounded.

After the call to **_ldfcvt()**, the position of the decimal point is available in *dpos*, and the sign of *value* is available in *sign*.

- *dpos* gives the position of the decimal point relative to the beginning of the string. Positive values indicate the number of digits to the right; negative or zero values indicate that the decimal point is to the left of the first digit.
- If *sign* is 0 (zero), the returned string represents a positive number. Otherwise, the number is negative.

Return Value

`_ldfcvt()` returns a pointer to a static string of digits. The character string is placed in a static buffer shared by [_ecvt\(\)](#) and [_ecvt\(\)](#), [_fcvt\(\)](#) and [_fcvt\(\)](#), [_ldecvt\(\)](#), and [_ldfcvt\(\)](#), and is overwritten by the next call to any of these functions.

See Also

[Conversion Functions](#) on page 425

ldiv()

Perform **long int** division with remainder

Targets

`ldiv()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */
ldiv_t ldiv(long int numerator, long int denominator);
```

Description

`ldiv()` computes the quotient and remainder of the division of *numerator* by *denominator* and stores them in a structure of type [ldiv_t](#). If the result cannot be represented, the behavior of `ldiv()` is undefined.

Return Value

`ldiv()` returns, in an [ldiv_t](#) structure, the values of the quotient and the remainder.

See Also

[div\(\)](#) — Perform integer division with remainder

[struct ldiv_t](#) — Result type of function `ldiv()`

Example

See the file `ldiv.c` in the *install_dir/docs/examples/* directory on your distribution.

lfind() and _lfind()

Search a linear list for a matching value, without modifying the list

Targets

Ifind() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_Ifind() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Ifind() is Non-ANSI; **_I**find() is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <search.h>           /* Required */
char *lfind(void *key, void *start, unsigned int *count,
            unsigned int width, int (*cmp)(const void *ptr1,
            const void *ptr2));
char *_lfind(void *key, void *start, unsigned int *count,
            unsigned int width, int (*cmp)(const void *ptr1,
            const void *ptr2));
```

Description

Ifind() and **_I**find() search an unordered linear list of data items for a match with a key value.

- *key* points to the key value.
- *start* points to the address of list.
- *count* points to the number of data items in the list.
- *width* is the width of each entry.

The match is determined when the compare function (**cmp()**) returns a 0 (zero) value. Function **cmp()** is a user-defined compare function that is called with two pointers (*ptr1* and *ptr2*) that must point to the *key* object and to an entry in the list, in that order.

- If **ptr1* is equal to **ptr2*, the **cmp()** function returns 0 (zero).
- If **ptr1* is not equal to **ptr2*, the **cmp()** function returns a non-zero **int** value.

If **cmp()** returns a non-zero value, **I**find() (or **_I**find()) adds *width* to the *start* address and again calls **cmp()** to compare *key* to this next entry in the list. This process continues until *count* elements have been compared or a match is found.

NOTE **I**find() and **_I**find() never modify the linear list.

Return Value

If a matching entry is found, **I**find() and **_I**find() return the address of that entry.

If no matching entry is found, **I**find() and **_I**find() return **NULL**.

See Also[bsearch\(\)](#) — Perform a binary search[lsearch\(\) and _lsearch\(\)](#) — Search a linear list for a matching value, modifying if necessary[qsort\(\)](#) — Perform a quicksort**Example**See the file `line.c` in the `install_dir/docs/examples/` directory on your distribution.

localeconv()

Query current locale for numeric format

Targets**localeconv()** works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <locale.h>          /* Required */
struct lconv *localeconv(void);
```

Description

localeconv() fills in a **struct [lconv](#)** with numeric formatting information appropriate for the current locale. The structure cannot be modified by the program but can be overwritten by subsequent calls to **localeconv()** or **setlocale()**.

Return Value**localeconv()** returns a pointer to an object of type **struct [lconv](#)**.**System issues**

On DOS and Windows NT, the native locale is the C locale and no other locales are supported by this implementation.

See Also[setlocale\(\)](#) — Set the current locale**struct [lconv](#)** — Locale-specific numeric representation**Example**See the file `localeco.c` in the `install_dir/docs/examples/` directory on your distribution.

localtime() and _localtime()

Convert a **time_t** value to a **struct tm**

Targets

`localtime()` and `_localtime()` work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`localtime()` is ANSI compliant; `_localtime()` is Extra-ANSI.

Reentrancy

Not-Reentrant

Synopsis

```
#include <time.h> /* Required */
struct tm *localtime(const time_t *timer);
struct tm *_localtime(const time_t *timer, struct tm *tmbuf)
```

Description

`localtime()` converts the time represented by *timer* into a **struct** containing the individual components of the time and date.

`_localtime()` is equivalent to `localtime()`, except `_localtime()` stores the result in structure *tmbuf*, as opposed to a static structure.

CAUTION The **tm** **struct** is overwritten by each call to `localtime()`.

Return Value

`localtime()` and `_localtime()` return a pointer to a **tm** **struct**.

System issues

On UNIX, `localtime()` and `_localtime()` depend on the environment variable TZ. If TZ is not set, the function defaults to Coordinated Universal Time (UTC).

See Also

[Time-Related Functions](#) on page 434

Type [**time_t**](#) — [Representation of a date and time](#)

[**struct tm**](#) — [Components of a date and time](#)

[Environment Variable TZ](#)

Example

See the file `asctime.c` in the `install_dir/docs/examples/` directory on your distribution.

log()

Calculate natural logarithm (base e)

Targets

`log()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
double log(double x);
```

Description

log() calculate the logarithm of x , base e. **log()** might be inlined when the toggle **Recognize_library** is On. Refer to the **MetaWare C/C++ Programmer's Guide** for details.

Return Value

log() returns the logarithm of x , base e. If $x \leq 0.0$, **log()** sets **errno** to [EDOM \(Mathematical domain error\)](#) and returns 0.0.

On UNIX, if $x == 0.0$, **log()** sets **errno** to [ERANGE \(Mathematical range error\)](#) and returns [HUGE_VAL](#).

See Also

[exp\(\)](#) — Calculate exponential (e to the power x)

[log10\(\)](#) — Calculate common logarithm (base 10)

[logf\(\)](#) — Calculate natural logarithm (base e) using single-precision floating point

Example

See the file `log.c` in the *install_dir/docs/examples/* directory on your distribution.

log10()

Calculate common logarithm (base 10)

Targets

log10() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
double log10(double x);
```

Description

log10() calculates the logarithm of x , base 10. **log10()** might be inlined when the toggle **Recognize_library** is On. Refer to the **MetaWare C/C++ Programmer's Guide** for details.

Return Value

log10() returns the logarithm of x , base 10. If $x \leq 0.0$, **log10()** sets **errno** to [EDOM \(Mathematical domain error\)](#) and returns 0.0.

On UNIX, if $x == 0.0$, **log10()** sets `errno` to [ERANGE \(Mathematical range error\)](#) and returns [HUGE_VAL](#).

See Also

[exp\(\)](#) — Calculate exponential (e to the power x)

[log\(\)](#) — Calculate natural logarithm (base e)

[log10f\(\)](#) — Calculate common logarithm (base 10) using single-precision floating point

Example

See the file `log10.c` in the `install_dir/docs/examples/` directory on your distribution.

log10f()

Calculate common logarithm (base 10) using single-precision floating point

Targets

log10f() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
float log10f(float x);
```

Description

log10f() calculates the logarithm of x , base 10. **log10f()** might be inlined when the toggle `Recognize_library` is On. Refer to the **MetaWare C/C++ Programmer's Guide** for details.

Return Value

log10f() returns the logarithm of x , base 10. If $x \leq 0.0$, **log10f()** sets `errno` to [EDOM \(Mathematical domain error\)](#) and returns 0.0.

On UNIX, if $x == 0.0$, **log10f()** sets `errno` to [ERANGE \(Mathematical range error\)](#) and returns [HUGE_VAL](#).

See Also

[exp\(\)](#) — Calculate exponential (e to the power x)

[expf\(\)](#) — Calculate exponential (e to the power x) using single-precision floating point

[log\(\)](#) — Calculate natural logarithm (base e)

[logf\(\)](#) — Calculate natural logarithm (base e) using single-precision floating point

log2() and _log2()

Calculate logarithm (base 2)

Targets

log2() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_log2() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

log2() is Non-ANSI; **_log2()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <math.h>           /* Required */
double log2(double x);
```

Description

log2() and **_log2()** calculate the logarithm of x , base 2.

Return Value

log2() returns the logarithm of x , base 2. If $x \leq 0.0$, **log2()** sets `errno` to [EDOM \(Mathematical domain error\)](#) and returns 0.0.

On UNIX, if $x == 0.0$, **log2()** sets `errno` to [ERANGE \(Mathematical range error\)](#) and returns [HUGE_VAL](#).

See Also

[log\(\)](#) — Calculate natural logarithm (base e)

[log10\(\)](#) — Calculate common logarithm (base 10)

logf()

Calculate natural logarithm (base e) using single-precision floating point

Targets

logf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
float logf(float x);
```

Description

logf() calculate the logarithm of x , base e. **logf()** might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

logf() returns the logarithm of x , base e. If $x \leq 0.0$, **logf()** sets `errno` to [EDOM](#) ([Mathematical domain error](#)) and returns 0.0.

On UNIX, if $x == 0.0$, **logf()** sets `errno` to [ERANGE](#) ([Mathematical range error](#)) and returns [HUGE_VAL](#).

See Also

[exp\(\)](#) — [Calculate exponential \(e to the power x\)](#)

[expf\(\)](#) — [Calculate exponential \(e to the power x\) using single-precision floating point](#)

[log\(\)](#) — [Calculate natural logarithm \(base e\)](#)

[log10\(\)](#) — [Calculate common logarithm \(base 10\)](#)

[log10f\(\)](#) — [Calculate common logarithm \(base 10\) using single-precision floating point](#)

longjmp()

Execute a non-local jump

Targets

longjmp() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <setjmp.h>           /* Required */  
void longjmp(jmp_buf env, int val);
```

Description

longjmp() restores (jumps to) the calling environment referenced by *env*.

env must have been initialized by a previous call to **setjmp(*env*)**. *env* references the calling environment existing at the point of the most recent call to **setjmp(*env*)**.

CAUTION If *env* has not been initialized by **setjmp()**, or if the routine in which the call to **setjmp()** occurred has returned before the supposedly corresponding call to **longjmp()**, the behavior is undefined. An uninitialized or dangling environment reference results in the program attempting to execute in an environment that is in a shambles.

Calling **longjmp()** has an indeterminate effect on any variables of storage-class **register** declared within the function containing the corresponding [setjmp\(\)](#), but has no effect on any other objects.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

longjmp() does not return in the classic sense. Completion of a **longjmp()** call appears to be a return of the corresponding call to **setjmp()** (that is, the most recent call to **setjmp()** with the same argument *env*).

Such a “return” from **longjmp()** (which appears to be a return from **setjmp()**) returns a non-zero value.

- If *val* is 0 (zero), the “return” value from **longjmp()** is 1 (one).
- If *val* is not 0 (zero), the “return” value from **longjmp()** is *val*.

These apparently equivalent kinds of “returns” from **setjmp()** can be distinguished, because the return value from an actual call to **setjmp()** is 0 (zero). (This protocol is patterned after that of the **fork** call of UNIX.)

See Also

[**setjmp\(\)** — Save a reference to the current calling environment for a subsequent non-local jump](#)

Example

See the file `longjmp.c` in the *install_dir/docs/examples/* directory on your distribution.

_lrotl() and _lrotr()

Rotate a **long int** left or right

Targets

The **_lrot***(*0*) functions work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Both functions are Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h> /* Required */
unsigned long _lrotl(unsigned long value, int bits);
unsigned long _lrotr(unsigned long value, int bits);
```

Description

_lrotl() rotates *value* to the left by the number of bits given in *bits*.

_lrotr() rotates *value* to the right by *bits* bits.

Return Value

_lrotl() returns *value* rotated to the left.

_lrotr() returns *value* rotated to the right.

See Also

[**_crotl\(\) and _crotr\(\)**, **_crol\(\)** and **_crotr\(\)** — Rotate a char left or right](#)

[rotl\(\) and rotr\(\)](#), [srotl\(\) and srotr\(\)](#) — Rotate an int left or right

[srotl\(\) and srotr\(\)](#), [srotl\(\) and srotr\(\)](#) — Rotate a short int left or right

Example

See the file `1rot.c` in the `install_dir/docs/examples/` directory on your distribution.

Isearch() and _lsearch()

Search a linear list for a matching value, modifying if necessary

Targets

Isearch() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_lsearch() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Isearch() is Non-ANSI; **_lsearch()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <search.h> /* Required */
char *lsearch(void *key, void *start, unsigned int *count,
              unsigned int width, int (*cmp)(const void *ptr1,
              const void *ptr2));
char *_lsearch(void *key, void *start, unsigned int *count,
               unsigned int width, int (*cmp)(const void *ptr1,
               const void *ptr2));
```

Description

Isearch() and **_lsearch()** search an unordered linear list of data items for a match with a key value.

- *key* points to the key value.
- *start* points to the address of list.
- *count* points to the number of data items in the list.
- *width* is the width of each entry.

The match is determined when the compare function (`cmp()`) returns a 0 (zero) value. Function `cmp()` is a user-defined compare function that is called with two pointers (`ptr1` and `ptr2`) that must point to the *key* object and to an entry in the list, in that order.

- If `*ptr1` is equal to `*ptr2`, the `cmp()` function returns 0 (zero).
- If `*ptr1` is not equal to `*ptr2`, the `cmp()` function returns a non-zero **int** value.

If `cmp()` returns a non-zero value, `lsearch()` or `_lsearch()` adds *width* to the *start* address and again calls `cmp()` to compare *key* to this next entry in the list. This continues until *count* elements have been compared or a match is found.

If `lsearch()` or `_lsearch()` cannot find *key* in the list (`cmp()` returns a non-zero value), they add *key* to the end of the list. Whenever a new list element is added, *count* is incremented by 1 (one).

Return Value

If a matching entry is found, `lsearch()` and `_lsearch()` return the address of that entry. If no matching entry is found, `lsearch()` and `_lsearch()` return the address of the entry that was added to the end of the list.

See Also

[bsearch\(\)](#) — Perform a binary search

[lfind\(\)](#) and [_lfind\(\)](#) — Search a linear list for a matching value, without modifying the list

[qsort\(\)](#) — Perform a quicksort

lseek() and _lseek()

Change the position in a file

Targets

`lseek()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_lseek()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`lseek()` is Non-ANSI; `_lseek()` is Extra-ANSI.

Reentrancy

`lseek()` is Not-Reentrant; `_lseek()` is Reentrant.

Synopsis

```
/* Required for DOS and Windows targets: */
#include <iо.h>
/* Required for all other targets: */
#include <unistd.h>

#include <stdio.h>
long lseek(int handle, long from, int where);
long _lseek(int handle, long from, int where);
```

Description

`lseek()` and `_lseek()` change the current position in the file specified by *handle*, thus changing the location next read from or written to. These functions use the value of the *where* argument to determine where to begin measuring the change of position. The *where* argument can have one of the following values:

[**SEEK_CUR**](#) — [Indicates the current position of the file pointer.](#)

[**SEEK_END**](#) — [Indicates the end of the file.](#)

[**SEEK_SET**](#) — [Indicates the beginning of the file.](#)

For more information about these constants, see [Constants and Globals](#) on page 319.

Iseek() and **_Iseek()** begin with the *where* position in the file, and go from that position the number of bytes indicated by the *from* argument.

Iseek() and **_Iseek()** have no effect on writing to a file opened in append mode.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

Upon normal completion, **Iseek()** and **_Iseek()** return the new position in the file, as measured from the beginning of the file. If an error occurs, **Iseek()** and **_Iseek()** return -1 and set **errno** to one of the following values:

[**EBADF**](#) — [“Bad” \(invalid or inaccessible\) file handle](#)

[**EINVAL**](#) — [Invalid argument given](#)

An error that sets **errno** to **EINVAL** can mean that the function attempted to move to an invalid position in the file.

On non-file-structured devices such as keyboards and screens, the return value is undefined.

See Also

[**fseek\(\)**](#) — [Move file pointer to new location in file](#)

[**tell\(\)** and **_tell\(\)**](#) — [Report the current position in a file](#)

Constants [**SEEK_CUR**](#) and [**SEEK_END**](#) and [**SEEK_SET**](#), [**SEEK_END**](#), and [**SEEK_SET**](#) —
Macros used to specify position in file

Example

See the file `lseek.c` in the *install_dir/docs/examples/* directory on your distribution.

Itoa() and _Itoa()

Convert a **long int** value to an ASCII string

Targets

Itoa() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_Itoa() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Itoa() is Non-ANSI; **_itoa()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h>      /* Required */
char *ltoa(long value, char *target, int radix);
char *_ltoa(long value, char *target, int radix);
```

Description

Itoa() and **_ltoa()** convert any **long** integer *value* into an ASCII string *target*. The conversion is performed according to the number system specified by *radix*, which can be any number between 2 and 36.

- If *radix* is the commonly used decimal 10, a negative *value* will have a leading minus sign; in addition, the maximum *target* is 12 bytes, consisting of an optional leading minus sign, 10 digits, and the final null byte.
- If *radix* is 2, the maximum size for *target* is 33 bytes, including the terminating null byte.

Return Value

Itoa() and **_ltoa()** return a pointer to the start of the result string.

See Also

[atol\(\)](#) — [Convert a prefix of a string to a long int](#)

[Conversion Functions](#) on page 425

Example

See the file `ltoa.c` in the *install_dir/docs/examples/* directory on your distribution.

_makepath()

Create a path string from components

Targets

_makepath() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>    /* Required */
void _makepath(char *path, char *drive,
               char *dirpath, char *filename, char *extension);
```

Description

_makepath() creates a string from a drive letter, directory path, file name, and file extension, returning the composite result in the buffer pointed to by *path*.

There are no size limits on the fields, but the composite returned in *path* must be no longer than **MAX_PATH** (as defined in `stdlib.h`), and *path* must point to a buffer large enough to hold the complete pathname.

The *drive*, *dirpath*, *filename*, and *extension* arguments point to buffers containing the following pathname elements:

- *drive* is a letter ('A,' 'B,' and so on) corresponding to the desired drive. The trailing colon, if omitted, is inserted by `_makepath()`. If *drive* is **NULL** or points to the empty string, no colon or drive specification appears in the path string.
- *dirpath* is the directory path, not including the drive specification or file name. Either forward or backward slashes can be used. The trailing slash, if omitted, is inserted by `_makepath()`. If *dirpath* is **NULL** or points to the empty string, no slash or directory path appears in the path string.
- *filename* is the base file name with no extensions. If *filename* is **NULL** or empty, no file appears in the path string.
- *extension* is the file-name extension. The period, if omitted, is inserted by `_makepath()`. If *extension* is **NULL** or points to the empty string, no period or file extension appears in the path string.

See Also

[File-Related Functions](#) on page 429

Constants [**MAX_*** — Maximum length of several pathname components](#)

Example

See the file `makepath.c` in the *install_dir/docs/examples/* directory on your distribution.

malloc()

Dynamically allocate uninitialized storage

Targets

`malloc()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>          /* Required */  
void *malloc(size_t size);
```

Description

`malloc()` dynamically allocates a block of uninitialized storage whose size is *size* bytes. To free up allocated storage no longer in use, call function [`free\(\)`](#).

NOTE To verify that the allocation request was successful, check for a non-**NULL** return value.

Return Value

malloc() returns a pointer to the lowest byte of the newly allocated storage. If the space cannot be allocated, **malloc()** returns **NULL**.

See Also

[Memory Allocation and Deallocation Functions](#) on page 434

Example

See the file `malloc.c` in the *install_dir/docs/examples/* directory on your distribution.

matherr(), matherrx() and _matherr(), _matherrx()

Provide standard error reporting for math functions

Targets

All the ***matherr***() functions work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The **matherr***() functions are Non-ANSI; the **_matherr***() functions are Extra-ANSI.

Reentrancy

All the ***matherr***() functions are Not-Reentrant.

Synopsis

```
#include <math.h> /* Required */
int matherr(struct exception *except);
int _matherr(struct exception *except);
int matherrx(struct exception *except);
int _matherrx(struct exception *except);
```

Description

As defined, every math function that detects an error places information about the error in a standard data structure (**struct exception**) and then calls **matherr()** (or **_matherr()**). What happens in **matherr()** (or **_matherr()**) is up to you, the user, so **matherr()** and **_matherr()** are actually *user-defined* functions.

The nature of the **exception** structure allows **matherr()** (or **_matherr()**) to reprocess the data that caused the error and provide a return value to be used by the calling function in place of the error-causing original condition.

If a user-defined version of **matherr()** (or **_matherr()**) returns a non-zero value, the library causes the math function to return the value of *except->retval* and not set global variable **errno**.

Member *except->retval* is initialized by the library to contain an appropriate default value prior to calling **matherr()** or **_matherr()**, but the function is free to change the value.

matherr() and **_matherr()** are provided as a service to programmers who want error messages printed to standard error when math errors occur. **matherr()** (and **_matherr()**, as provided in the MetaWare C distribution, do the following:

- Print a description of the error to `stderr`.
- Set global variable `errno` to a default value.
- Return 1 (one).

Member `except->retval` is not altered, so the default value determined by the library is used.

The MetaWare C library provides default versions of `matherr()` and `_matherr()` that return 0 (zero) if an error occurs. When `matherr()` (or `_matherr()`) returns 0 (zero), the library sets `errno` to a default value determined by the nature of the error, and causes the math function that experienced the error to return a default value.

User-Defined Math Error Handlers

An application supplies its own `matherr()` (or `_matherr()`) by linking a user-defined version of `matherrx()` (or `_matherrx()`) instead of pulling in the default version from the library. Alternatively, the math error handler can be changed dynamically (and repeatedly) at run time with the `_set_matherr()` function.

CAUTION Applications that use `_set_matherr()` should not also supply their own `matherr()` (or `_matherr()`) function.

Return Value

On success, `matherr()` and `_matherr()` return a non-zero value. On failure, `matherr()` and `_matherr()` return 0 (zero).

`matherrx()` and `_matherrx()` always return 1 (one).

See Also

[_set_matherr\(\)](#) — User defined math error function

[struct exception](#) — Nature of a math error

Global variable `errno` — [An int used to record error numbers](#)

_max()

Determine maximum of any arithmetic type

Targets

`_max()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

`_max(e1, e2, ..., en)`

Description

Compiler intrinsic `_max()` determines the maximum of any arithmetic type. Each argument ($e_1, e_2, e_3, \dots, e_n$) must be an arithmetic type. The result is the maximum of ($e_1, e_2, e_3, \dots, e_n$), and the type of the result is the “common” type of the arguments.

The operands of `_max()` are evaluated at most once, unlike the standard macro definition of `_max()`:

```
#define max(x,y) ((x) > (y) ? (x) : (y))
```

where one argument is evaluated once and the other twice.

Return Value

`_max()` returns the maximum number that is passed to it. The type of the return value is the common type of the arguments to `_max()`. (See the example for a demonstration.)

See Also

[min\(\)](#) — [Determine minimum of any arithmetic type](#)

Example

See the file `max.c` in the `install_dir/docs/examples/` directory on your distribution.

mblen()

Determine length of a multibyte character

Targets

`mblen()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */
int mblen(const char *s, size_t n);
```

Description

If s is not `NULL`, `mblen()` determines the length of the multibyte character pointed to by s .

Return Value

When the next n bytes pointed to by s form a valid multibyte character, `mblen()` returns the number of bytes that make up the converted multibyte character.

When the next n or fewer bytes pointed to by s do not form a valid multibyte character, `mblen()` returns `-1`.

If s is `NULL` and multibyte characters have state-dependent encodings, `mblen()` returns a non-zero value. If s is `NULL` and multibyte characters do not have state-dependent encodings, `mblen()` returns 0 (zero).

See Also

[mbstowcs\(\)](#) — [Convert a multibyte string to wchar_t array](#)

[**mbtowc\(\)**](#) — Convert a multibyte character to wchar_t code

[**wcstombs\(\)**](#) — Convert a wchar_t array to a multibyte string

[**wctomb\(\)**](#) — Convert a wchar_t code to a multibyte character

Type [**wchar_t**](#) — Integral type; extended-character-set values

Constant [**MB_CUR_MAX**](#) — Maximum number of bytes for a character in current locale

Constant [**MB_LEN_MAX**](#) — Maximum number of bytes in a multibyte character

mbstowcs()

Convert a multibyte string to wchar_t array

Targets

mbstowcs() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
```

Description

mbstowcs() converts the multibyte string *s*, which begins in the initial shift state, into an array of values of type [**wchar_t**](#), pointed to by *pwcs*, until *n* values are stored or a value of 0 (zero) is stored.

CAUTION If **mbstowcs()** stores *n* bytes before a value of 0 (zero) is stored, the destination array will not be terminated with a 0 (zero) value.

Return Value

On success, **mbstowcs()** returns the number of bytes stored.

If an invalid multibyte character is encountered, **mbstowcs()** returns -1 converted to [**size_t**](#).

See Also

[**mblen\(\)**](#) — Determine length of a multibyte character

[**mbtowc\(\)**](#) — Convert a multibyte character to wchar_t code

[**wcstombs\(\)**](#) — Convert a wchar_t array to a multibyte string

[**wctomb\(\)**](#) — Convert a wchar_t code to a multibyte character

Type [**wchar_t**](#) — Integral type; extended-character-set values

Constant [**MB_CUR_MAX**](#) — Maximum number of bytes for a character in current locale

Constant [**MB_LEN_MAX**](#) — Maximum number of bytes in a multibyte character

mbtowc()

Convert a multibyte character to **wchar_t** code

Targets

mbtowc() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */
int mbtowc(wchar_t *pwc, const char *s, size_t n);
```

Description

If *s* is not **NULL**, **mbtowc()** determines the length of the multibyte character pointed to by *s*. Then it determines the value of type **wchar_t** that corresponds to that multibyte character. If *s* points to a valid multibyte character and *pwc* is not **NULL**, the value is stored in the object pointed to by *pwc*.

Return Value

When *s* points to a valid multibyte character, **mbtowc()** returns the number of bytes that comprise the converted multibyte character. At most, **mbtowc()** will examine *n* characters, and the return value will never exceed *n* or the value of [**MB_CUR_MAX**](#).

When *s* points to an invalid multibyte character, **mbtowc()** returns -1.

If *s* is **NULL** and multibyte characters have state-dependent encodings, **mbtowc()** returns a non-zero value. If *s* is **NULL** and multibyte characters do not have state-dependent encodings, **mbtowc()** returns 0 (zero).

See Also

[**mblen\(\)** — Determine length of a multibyte character](#)

[**mbstowcs\(\)** — Convert a multibyte string to wchar_t array](#)

[**wctombs\(\)** — Convert a wchar_t array to a multibyte string](#)

[**wctomb\(\)** — Convert a wchar_t code to a multibyte character](#)

Type [**wchar_t** — Integral type; extended-character-set values](#)

Constant [**MB_CUR_MAX** — Maximum number of bytes for a character in current locale](#)

Constant [**MB_LEN_MAX** — Maximum number of bytes in a multibyte character](#)

memccpy() and _memccpy()

Copy memory until count or character

Targets

`memccpy()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_memccpy()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`memccpy()` is Non-ANSI; `_memccpy()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>
void *memccpy(void *target, void *source, int c, unsigned count);
void *_memccpy(void *target, void *source, int c, unsigned count);
```

Description

`memccpy()` and `_memccpy()` copy bytes from *source* to *target*, until either the character *c* has been copied, or *count* bytes have been copied.

Return Value

`memccpy()` and `_memccpy()` return **NULL** if the character *c* was not found. If the character *c* was found, `memccpy()` and `_memccpy()` return a pointer to the *target* byte following the character *c*.

See Also

[memchr\(\)](#) — Find a character in an area of memory

[memcmp\(\)](#) — Compare two areas of memory

[memcpy\(\)](#) — Copy from one place in memory to another

[memset\(\)](#) — Duplicate a character across an area of memory

Example

See the file `memccpy.c` in the *install_dir/docs/examples/* directory on your distribution.

memchr()

Find a character in an area of memory

Targets

`memchr()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */
void *memchr(const void *s, int c, size_t n);
```

Description

memchr() finds the first occurrence of the character *c* (the least significant byte of **int** *c*) in the *n* bytes starting at the location referenced by *s*.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

memchr() returns a pointer to the character if it is found. It returns **NULL** if the character is not found.

Surprises

memchr() takes *c* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to **memchr()**, it is automatically coerced to type **int**.

See Also

[findclose\(\) and _findfirst\(\) and _findnext\(\)](#)

[String-Handling Functions](#) on page 432

Example

See the file `memchr.c` in the *install_dir/docs/examples/* directory on your distribution.

memcmp()

Compare two areas of memory

Targets

memcmp() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */
int memcmp(const void *ptr1, const void *ptr2, size_t count);
```

Description

memcmp() compares two areas of memory, starting at *ptr1* and *ptr2*. The bytes are treated as characters — the comparison is lexicographical. If the characters are equal, **memcmp()** continues with the next byte, until it finds a difference between two bytes or until *count* bytes have been compared, whichever occurs first.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

This is the result of the comparison:

- If a byte in *ptr1* compares less than the equivalent byte in *ptr2*, **memcmp()** returns an **int** less than 0 (zero).
- If a byte in *ptr1* compares greater than the equivalent byte in *ptr2*, **memcmp()** returns an **int** greater than 0 (zero).
- If the first *count* bytes in *ptr1* compare equal to the first *count* bytes in *ptr2*, **memcmp()** returns **int** 0 (zero).

See Also

[**memicmp\(\)** and **_memicmp\(\)** — Compare bytes ignoring case](#)

[**strcmp\(\)** — Compare one string to another](#)

[**strncmp\(\)** — Compare characters of one string to characters of another](#)

Example

See the file `memcmp.c` in the *install_dir/docs/examples/* directory on your distribution.

memcpy()

Copy from one place in memory to another

Targets

memcpy() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */  
void *memcpy(void *dest, const void *source, size_t n);
```

Description

memcpy() copies *n* bytes from the location referenced by *source* to the location referenced by *dest*. **memcpy()** copies from left to right (from low addresses to high addresses). **memcpy()** might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

If *source* < *dest* < (*source* + *n*) — that is, if the source and destination areas overlap, with the destination area beginning within the source area — the behavior is undefined; use **_rmemcpy()**.

CAUTION `memcpy()` writes on *n* bytes.

Return Value

`memcpy()` returns a pointer to *dest*.

Surprises

`memcpy()` copies the second argument to the first.

See Also

[`memset\(\)`](#) — [Duplicate a character across an area of memory](#)

[`rmemcpy\(\)`](#) — [Copy from one place in memory to another](#)

[`rstrcpy\(\)`](#) — [Copy one string onto another](#)

[`rstrncpy\(\)`](#) — [Copy a number of characters from one string into another](#)

[`strcpy\(\)`](#) — [Copy the characters of one string into another](#)

[`strncpy\(\)`](#) — [Copy a number of characters from one string into another](#)

Example

See the file `memcpy.c` in the *install_dir/docs/examples/* directory on your distribution.

memicmp() and _memicmp()

Compare bytes ignoring case

Targets

`memicmp()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_memicmp()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`memicmp()` is Non-ANSI; `_memicmp()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>
int memicmp(void *ptr1, void *ptr2, unsigned count);
int _memicmp(void *ptr1, void *ptr2, unsigned count);
```

Description

`memicmp()` and `_memicmp()` compare two areas of memory, starting at *ptr1* and *ptr2*. If the characters are equal, `memicmp()` and `_memicmp()` continue with the next byte, until they find a difference between two bytes or until *count* bytes have been compared, whichever occurs first. The

comparison is performed without regard to case. Each character is converted to lowercase in a temporary area, and then compared.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

This is the result of the comparison:

- If a byte in *ptr1* compares less than the equivalent byte in *ptr2*, **memicmp()** and **_memicmp()** return an **int** less than 0 (zero).
- If a byte in *ptr1* compares greater than the equivalent byte in *ptr2*, **memicmp()** and **_memicmp()** return an **int** greater than 0 (zero).
- If the first *count* bytes in *ptr1* compare equal to the first *count* bytes in *ptr2*, **memicmp()** and **_memicmp()** return **int** 0 (zero).

See Also

[**memccpy\(\)** and **_memccpy\(\)**](#) — [Copy memory until count or character](#)

[**memchr\(\)**](#) — [Find a character in an area of memory](#)

[**memcmp\(\)**](#) — [Compare two areas of memory](#)

[**memcpy\(\)**](#) — [Copy from one place in memory to another](#)

[**memset\(\)**](#) — [Duplicate a character across an area of memory](#)

Example

See the file `memicmp.c` in the `install_dir/docs/examples/` directory on your distribution.

memmove()

Copy memory nondestructively

Targets

memmove() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */  
void *memmove(void *dest, const void *source, size_t n);
```

Description

memmove() copies *n* bytes from the location referenced by *source* to the location referenced by *dest*. The copy will be done correctly even if the source and destination overlap.

Return Value

memmove() returns a pointer to *dest*.

Surprises

memmove() copies the second argument to the first.

See Also

[memcpy\(\)](#) — Copy from one place in memory to another

[rmemcpy\(\)](#) — Copy from one place in memory to another

memset()

Duplicate a character across an area of memory

Targets

memset() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */
void *memset(void *start, int fill, size_t len);
```

Description

memset() copies the character *fill* (the least significant byte of **int** *fill*) into each of *len* bytes starting at the location pointed to by *start*. **memset()** might be inlined when toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

CAUTION **memset()** writes on *n* bytes.

Return Value

memset() returns a pointer to *start*.

Surprises

memset() copies the second argument to the first.

memset() takes *fill* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to **memset()**, it is automatically coerced to type **int**.

See Also

[memcpy\(\)](#) — Copy from one place in memory to another

[strcpy\(\)](#) — Copy the characters of one string into another

[strncpy\(\)](#) — Copy a number of characters from one string into another

Example

See the file `memset.c` in the `install_dir/docs/examples/` directory on your distribution.

_min()

Determine minimum of any arithmetic type

Targets

`_min()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

`_min(e1, e2, ..., en)`

Description

Compiler intrinsic `_min()` determines the minimum of any arithmetic type. Each function argument (*e*₁, *e*₂, *e*₃, ..., *e*_{*n*}) must be an arithmetic type. The result is the minimum of (*e*₁, *e*₂, *e*₃, ..., *e*_{*n*}), and the type of the result is the “common” type of the arguments.

The operands of `_min()` are evaluated at most once, unlike the standard macro definition of `min`:

```
#define min(x,y) ((x) < (y) ? (x) : (y))
```

where one argument is evaluated once and the other twice.

Return Value

`_min()` returns the minimum number that is passed to it. The type of the return value is the common type of the arguments to `_min()`. (See the example for a demonstration.)

See Also

[max\(\) — Determine maximum of any arithmetic type](#)

Example

See the file `max.c` in the `install_dir/docs/examples/` directory on your distribution.

mkdir() and _mkdir()

Create a new directory

Targets

`mkdir()` and `_mkdir()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`mkdir()` is Non-ANSI; `_mkdir()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <direct.h>           /* Required */
int mkdir (char *path);
int _mkdir (char *path);
```

Description

mkdir() and **_mkdir()** create a new directory specified by *path*. Each component of *path*, except the last, must reference an existing directory. To create a multilevel directory structure, create each component with a separate call to **mkdir()**; for example:

```
mkdir("c:\\top");
mkdir("c:\\top\\next");
mkdir("c:\\top\\next\\bottom");
```

If directory *c:\top\next* already exists, call **mkdir()** as follows:

```
mkdir("c:\\top\\next\\bottom");
```

Return Value

mkdir() and **_mkdir()** return 0 (zero) if the new directory was created. If the new directory was not created, **mkdir()** and **_mkdir()** return -1 and set *errno* to one of the following values:

EACCES — [Access error; permission denied](#)

ENOENT — [File or directory not found; pathname element does not exist](#)

An error that sets *errno* to **EACCESS** means the directory was not created; *path* refers to an existing directory.

See Also

[chdir\(\) and _chdir\(\)](#) — [Change the current working directory](#)

[getcwd\(\) and _getcwd\(\)](#) — [Get full pathname of the current working directory](#)

[rmdir\(\) and _rmdir\(\)](#) — [Remove a directory](#)

Global variable [errno](#) — [An int used to record error numbers](#)

Example

See the file *mkdir.c* in the *install_dir/docs/examples/* directory on your distribution.

mktemp() and _mktemp()

Create a unique file name (does not create or open files)

Targets

mktemp() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_mktemp() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

mktemp() is Non-ANSI; **_mktemp()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>
char *mktemp (char *template);
```

Description

mktemp() and **_mktemp()** produce a unique file name by operating on a *template* argument of this form:

mkbaseXXXXXX

The *mkbase* component of *template* is a user-defined sequence of zero to six characters. **mktemp()** and **_mktemp()** replace the six ‘X’ characters of the template with *nn.ppp*, where *nn* is a base-36 number that is incremented with each call, and *ppp* is a base-36 encoding of the process ID.

Upon repeated calls, **mktemp()** and **_mktemp()** check previously returned names to see if files exist for those names. If the files exist (none of the previously returned names are available for use), the functions create a new name by incrementing the *nn* in the template.

For example, given a *template* argument of 102052XXXXXX, **mktemp()** and **_mktemp()** could produce the following sequence of unique file names:

```
102052z9.kmc
102052za.kmc
102052zb.kmc
```

Return Value

If the *template* argument is correctly formed and a unique file name is created, **mktemp()** and **_mktemp()** return a pointer to the new file-name string.

mktemp() and **_mktemp()** return **NULL** if unable to create a unique file name or if the *template* argument is incorrectly formed.

See Also

[fopen\(\)](#) — [Open a file](#)

[getpid\(\) and _getpid\(\)](#) — [Get unique ID of the calling process](#)

[open\(\) and _open\(\)](#) — [Open a file handle with a pathname](#)

Example

See the file `mktemp.c` in the *install_dir/docs/examples/* directory on your distribution.

mkttime()

Convert a **struct tm** to a **time_t** value

Targets

mkttime() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <time.h>          /* Required */
time_t mktime(struct tm *tptr);
```

Description

mktime() converts a local time, as described in the [tm](#) structure pointed to by *tptr*, to an equivalent time of the form returned by the **time()** function.

The values in the [tm](#) structure should be correct, but are not required to be within normal ranges. The [tm_wday](#) and [tm_yday](#) fields are ignored. The time is calculated, and then **localtime()** is called to set the fields of the [tm](#) structure to correct values.

The **mktime()** function depends on environment variable **TZ**. If **TZ** is not set, the function defaults to Coordinated Universal Time (UTC).

Return Value

mktime() returns the calendar time equivalent to the local time given in the [tm](#) structure.

See Also

[Time-Related Functions](#) on page 434

Type [time_t](#) — [Representation of a date and time](#)

[struct tm](#) — [Components of a date and time](#)

Example

See the file `mktm.c` in the *install_dir/docs/examples/* directory on your distribution.

modf()

Break a **double** into integer and fractional parts

Targets

modf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>          /* Required */
double modf(double value, double *iptr);
```

Description

modf() breaks *value* into integer and fractional parts. The integral part of *value* is stored in the object pointed to by *iptr*.

Return Value

modf() returns the fractional part of *value*.

See Also

[frexp\(\)](#) — Break a double into fraction and exponent

[frexpf\(\)](#) — Break a float into fraction and exponent

[modf\(\)](#) — Break a float into integer and fractional parts

Example

See the file `modf.c` in the *install_dir/docs/examples/* directory on your distribution.

modff()

Break a **float** into integer and fractional parts

Targets

modff() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
float modff(float value, float *iptr);
```

Description

modff() breaks *value* into integer and fractional parts. The integral part of *value* is stored in the object pointed to by *iptr*.

Return Value

modff() returns the fractional part of *value*.

See Also

[frexp\(\)](#) — Break a double into fraction and exponent

[frexpf\(\)](#) — Break a float into fraction and exponent

[modf\(\)](#) — Break a double into integer and fractional parts

_msize()

Get the size of a memory block in the heap

Targets

_msize() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <malloc.h>
size_t _msize(void *mblock);
```

Description

`_msize()` determines the size of the memory block pointed to by *mblock*. *mblock* must be a memory block allocated in the heap by a call to `malloc()`.

Return Value

`_msize()` returns a `size_t` value representing the number of bytes in *mblock*.

See Also

[malloc\(\)](#) — [Dynamically allocate uninitialized storage](#)

Type [size_t](#) — [Type of the result of sizeof\(\)](#)

Example

See the file `msize.c` in the *install_dir/docs/examples/* directory on your distribution.

offsetof()

Determine offset of a structure member

Targets

`offsetof` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stddef.h>    /* Required */
size_t offsetof(type, identifier)
```

Description

`offsetof` determines the offset of a structure member within a structure.

- *type* is the type of the structure.
- *identifier* is the name of the member.

The result is the offset in bytes of the member from the start of the structure. If the specified member is a bit field, the behavior is undefined.

Note `offsetof` is provided only as a macro.

Return Value

`offsetof` returns a `size_t` value indicating the offset described.

open() and _open()

Open a file handle with a pathname

Targets

`open()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

`_open()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`open()` is Non-ANSI; `_open()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <fcntl.h>
#include <types.h>
#include <stat.h>
#include <io.h>          /* Required for DOS and Windows targets */
#include <unistd.h>      /* Required for all other targets */

/* For existing files */
int open(char *pathname, int flags);
int _open(char *pathname, int flags);

/* For new files */
int open(char *pathname, int flags, int mode);
int _open(char *pathname, int flags, int mode);
```

Description

`open()` and `_open()`, called with three arguments, open existing files. Called with the additional *mode* argument, these functions create new files.

These functions can open a file for reading, writing, or both. Before setting the newly created file's permission, `open()` and `_open()` apply to the *mode* the current file-permission mask as set by [umask\(\)](#) and [umask\(\)](#).

The flags Parameter

The opening actions are controlled by these [O_* and O_*](#) and [O_* and O_*](#) flags, which are defined in `fcntl.h`:

[O_APPEND](#) or [O_APPEND](#) — Opens file for append

[O_CREAT](#) or [O_CREAT](#) — Creates file

[O_EXCL](#) or [O_EXCL](#) — Returns error if file exists

[**O_TRUNC**](#) or [**O_TRUNC**](#) — Destroys existing file before (re)creating

[**O_RDONLY**](#) or [**O_RDONLY**](#) — Opens read-only file

[**O_RDWR**](#) or [**O_RDWR**](#) — Opens file for reading and writing

[**O_WRONLY**](#) or [**O_WRONLY**](#) — Opens file for writing only

[**O_BINARY**](#) or [**O_BINARY**](#) — Opens file for binary data

[**O_TEXT**](#) or [**O_TEXT**](#) — Opens file with text translation enabled

Compatible flags can be joined together with the OR operator (|).

For more information about the [**O_*** and **O_***](#) and [**O_*** and **O_***](#) constants, see [Constants and Globals](#) on page 319

NOTE Use exactly one (no more and no less) of [**O_RDONLY**](#), [**O_RDWR**](#), and [**O_WRONLY**](#) (or [**_O_RDONLY**](#), [**_O_RDWR**](#), and [**_O_WRONLY**](#)).

CAUTION Use the [**O_TRUNC**](#) or [**_O_TRUNC**](#) flag with care, because it destroys all existing data in a file at the time of the open request.

A call to `open()` with flags [**O_CREATE**](#) and [**O_TRUNC**](#) (or a call to `_open()` with [**_O_CREATE**](#) and [**_O_TRUNC**](#)) produces the same result as a call to function [**creat\(\)** and **_creat\(\)**](#) or [**creat\(\)** and **_creat\(\)**](#).

The mode Parameter

The [**O_CREAT**](#) (or [**_O_CREAT**](#)) flag specifies that a new file might be created. The new file will have attributes set according to the *mode* parameter; allowable values for *mode*, defined in `stat.h`, are [**S_IREAD**](#), [**S_IWRITE**](#), and ([**S_IWRITE**](#) | [**S_IREAD**](#)).

For more information about the [**S_I***](#) constants, see [Constants and Globals](#) on page 319.

The file's *mode* takes effect when it is closed for the first time. Any opens after that first closing are controlled by the permissions given in *mode*.

System issues

On DOS and Windows NT, all files are readable (it is not possible to give write-only permission), so the read permission is included for compatibility with other systems, but is ignored here.

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

When a file is opened successfully, `open()` and `_open()` return a file handle. If an error occurs, `open()` and `_open()` return -1 and set global variable `errno` to one of the following values:

[**EACCES**](#) — [Access error; permission denied](#)

[**EEXIST**](#) — [File already exists, so it cannot be created](#)

[**EINVAL**](#) — [Invalid argument given](#)

[**EMFILE**](#) — [Too many open file handles](#)

[**ENOENT**](#) — [File or directory not found; pathname element does not exist](#)

An error that sets `errno` to **EACCES** means that access is not allowed to the specified *pathname*. The path might specify a read-only file or a directory, or a sharing violation occurred on a network system.

See Also

[File-Related Functions](#) on page 429

Constants [O_* and O_* and O_* and O_*](#) — [File-open mode flags](#)

Constants [S_I*](#) — [File permissions](#)

Global variable [errno](#) — [An int used to record error numbers](#)

Example

See the file `open.c` in the `install_dir/docs/examples/` directory on your distribution.

_opendir()

Open a POSIX directory stream

Targets

`_opendir()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <dirent.h>          /* Required */  
DIR *_opendir(const char *dirname)
```

Description

`_opendir()` opens directory stream *dirname* for reading.

The stream is set to read the first entry in the directory. The stream is represented by the [DIR](#) structure `_DIR` as defined in `dirent.h`. The fields of `_DIR` are not accessible to the user.

After a successful call to `_opendir()`, use `_readdir()` to read the set of file names in the directory. Use `_rewinddir()` to set the stream back to the first entry in the directory, and `_closedir()` to close the directory stream.

Return Value

On success, `_opendir()` returns a pointer to a directory stream. On failure, it returns **NULL** and sets global variable `errno` to one of the following values:

[ENOENT](#) — [File or directory not found; pathname element does not exist](#)

[ENOMEM](#) — [Not enough memory, or invalid DOS arena headers](#)

See Also

[closedir\(\)](#) — [Close a POSIX directory stream](#)

[readdir\(\)](#) — [Read a POSIX directory stream](#)

[rewinddir\(\)](#) — [Reset a POSIX directory stream](#)

struct [_dirent](#) — A POSIX file name for [_readdir\(\)](#)

Example

See the file `opendir.c` in the `install_dir/docs/examples/` directory on your distribution.

perror()

Print an error message

Targets

`perror()` works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
void perror(const char *str);
```

Description

`perror()` prints the following to the standard error file:

str: Error_message_string nwln

Error_message_string is an error-message string associated with the current value of global variable `errno`, and *nwln* is a newline.

If *str* is [NULL](#), or if the first character of *str* is the NUL character ('\0'), `perror()` prints only *Error_message_string* and the newline.

See Also

Global variable [errno](#) — [An int used to record error numbers](#)

Example

See the file `perror.c` in the `install_dir/docs/examples/` directory on your distribution.

pow()

Raise a **double** to a power

Targets

`pow()` works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>          /* Required */  
double pow(double x, double y);
```

Description

pow() calculates the value of x^y .

Return Value

pow() returns x raised to the power y .

- If $x = 0.0$ and $y \leq 0.0$, **pow()** sets global variable `errno` to **EDOM** and returns 0.0.
- If $x < 0.0$ and y is not an integral number, **pow()** sets global variable `errno` to **EDOM** and returns 0.0.
- If $|xy| > \text{HUGE_VAL}$, the return value of **pow()** has magnitude **HUGE_VAL** and has the same sign as x^y .

See Also

[exp\(\)](#) — Calculate exponential (e to the power x)

[expf\(\)](#) — Calculate exponential (e to the power x) using single-precision floating point

[lDEXP\(\)](#) — Multiply by a power of 2

[lDEXPF\(\)](#) — Multiply by a power of 2 using single-precision floating point

[POWF\(\)](#) — Raise a float to a power

POWF()

Raise a **float** to a power

Targets

POWF() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>          /* Required */  
float powf(float x, float y);
```

Description

POWF() calculates the value of x^y .

Return Value

POWF() returns x raised to the power y .

- If $x = 0.0$ and $y \leq 0.0$, **powf()** sets global variable `errno` to **EDOM** and returns 0.0.
- If $x < 0.0$ and y is not an integral number, **powf()** sets global variable `errno` to **EDOM** and returns 0.0.
- If $|xy| > \text{HUGE_VAL}$, the return value of **powf()** has magnitude **HUGE_VAL** and has the same sign as x^y .

See Also[exp\(\)](#) — Calculate exponential (e to the power x)[expf\(\)](#) — Calculate exponential (e to the power x) using single-precision floating point[ldexp\(\)](#) — Multiply by a power of 2[ldexpf\(\)](#) — Multiply by a power of 2 using single-precision floating point[pow\(\)](#) — Raise a double to a power

printf()

Print to `stdout`**Targets**`printf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

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

Description`printf()` writes to `stdout` according to the format string *format*.The discussion of the format string and conversion specifications in this section applies to `*printf()`, all the functions in the `printf()` family:[bprintf\(\)](#) — Print to a string of given size[cprintf\(\) and cprintf\(\)](#) — Print to screen[fprintf\(\)](#) — Print to a file[printf\(\)](#) — Print to `stdout`[sprintf\(\)](#) — Print to a string[vbprintf\(\)](#) — Print to a string of given size using var-arg macros[vfprintf\(\)](#) — Print to a file using var-arg macros[vprintf\(\)](#) — Print to `stdout` using var-arg macros

[vsprintf\(\)](#) — Print to a string using var-arg macros

CAUTION Garbage might be printed if the number of arguments passed to **printf()** is less than the number specified by *format*, or if the types of the arguments to **printf()** do not match the types specified by *format*.

Format String

format is a string containing conversion specifications that specify how to represent subsequent arguments in print. Any characters that are not elements of conversion specifications are simply printed out unchanged.

The representation of each argument following *format* is printed out at the point in the string where its conversion specification appears. The conversion specifications act as place holders for a printable representation of the corresponding argument. *format* is printed with each specification replaced by the representation of its argument.

If a conversion specification is not valid (that is, if the conversion character is not one of those listed in the section [Conversion Specifications](#)), global variable `errno` is set and that conversion is ignored. The argument is converted instead by the next conversion specification.

If the arguments do not match those specified by *format*, the behavior of **printf()** is undefined.

NOTE Too few arguments:

If the number of actual arguments is less than the number of arguments specified by *format*, ***printf()** moves through memory in undefined fashion, printing whatever it finds in the presumed argument list according to the specification in the format string.

Too many arguments:

If the number of arguments passed in is greater than the number specified by *format*, the excess arguments are evaluated by the standard function-call mechanism, but otherwise are ignored.

Conversion Specifications

This is the form of a conversion specification:

`'%' Flag* Field_width? Precision? Size? C_char`

The components of the conversion specification have the following meanings:

<u>Flag</u>	= '-' '+' ' ' '#' '0';
<u>Field_width</u>	= ((Digit - '0') Digits) '*';
<u>Precision</u>	= '.' (Digits '*');
<u>Size</u>	= 'h' 'l' 'L';
<u>C_char</u>	= 'd' 'i' 'o' 'u' 'x' 'X' 'f' 'e' 'E' 'g' 'G' 'c' 's' 'p' 'n' '%';

The '%' character sets off a conversion specification from ordinary characters.

(See [Regular Expressions](#) on page 18 for an explanation of this notation.)

The Flag Characters

The flag characters have the following meanings:

Flag '-'

The result of the conversion is left-justified within the field. Any padding appears on the right. If '-' does not appear, the result is right-justified.

Flags '+' and ' ' (space)

The result of a signed conversion begins with a plus, space, or minus sign.

- Negative values are printed beginning with a minus sign
- Given '+', positive values are printed beginning with a plus sign.
- Given ' ', positive values are printed beginning with a space.
- If both the ' ' flag and the '+' flag appear, the ' ' flag is ignored.
- If neither '+' nor ' ' appears, negative values begin with a minus sign, and positive values begin with the first digit of the result.

Flag '#'

The result is to be converted to an alternate form, specified in the description of each conversion character in the section [The C_char Specification](#). If no alternate form is mentioned in the description, the flag has no effect on that conversion.

Flag '0'

Padding is by zeros. If the '0' flag does not appear, padding is by spaces.

- If padding is by zeros, the sign (or space, if the ' ' flag appears), if any, precedes the zeros.
- If padding is by spaces, the spaces precede the sign.
- If the result is left justified ('-' appears), padding is by spaces on the right and the '0' flag has no effect.

The Field_width Specification

Each conversion takes place within a field of characters. The minimum size of the field can be specified by **Field_width**.

- If no **Field_width** appears, the field is the size of the result of the conversion.
- If **Field_width** is not an integer but is '*', the value of **Field_width** is taken from an **int** argument that precedes the argument to be converted.
- If the result of the conversion contains fewer characters than **Field_width** specifies, the field is padded with spaces or zeros.
- If there are more characters resulting from a conversion than specified by **Field_width**, the field is expanded so that they all get printed; **Field_width** never causes truncation.

The Precision Specification

The meaning of **Precision** varies from one conversion type to the next. Usually it specifies a maximum or minimum number of significant characters in the result. (For details, see the section [The C_char Specification](#), following.)

Precision differs from **Field_width** in that **Field_width** can cause padding, but can never affect the “value” of the result, while **Precision** affects the characters produced by converting an argument. For example, **Precision** can cause string truncation, or affect the number of characters that appear after the decimal point in a **double** conversion.

-
- If **Precision** is not an integer but is '*', the value of **Precision** is taken from an **int** argument that precedes the argument to be converted.
 - If both **Field_width** and **Precision** are specified by asterisks, the **Field_width** argument comes first, then the **Precision** argument, then the argument to be converted.
 - A negative **Precision** is taken as if **Precision** were missing.

In situations where **Precision** is 0 (zero) and the converted value consists of no characters (all except %E, %e, and %f conversions), the result is independent of any padding specified by **Field_width**.

The Size Specification

Size specifies the size of an argument whose type comes in more than one size; for example, **int** versus **long int**. In the section [The C_char Specification](#), the **Size** specification is mentioned in the description of each conversion that it can affect. If **Size** is specified for a conversion that it cannot affect, it is ignored.

The size characters have the following meanings:

- If **Size** is **h**, the argument is a **short int**.
- If **Size** is **l**, the argument is a **long int**.
- If **Size** is **L** or **ll**, the argument is a **long double** or a **long long**.

The C_char Specification

C_char specifies the type of conversion: both the type of the argument to be converted and the format of the converted result (modified by [Flag](#), [Field_width](#), [Precision](#), and [Size](#)).

- [“%” Conversion \(Plain ‘%’\)](#)
- [‘c’ Conversion \(Character As Integer\)](#)
- [‘d’ Conversion \(Signed Decimal Integer\)](#)
- [‘e’ or ‘E’ Conversion \(Scientific Notation Double or Float\)](#)
- [‘f’ Conversion \(Decimal Double Or Float\)](#)
- [‘g’ or ‘G’ Conversion \(Decimal Double or Float as ‘e’ or ‘f’\)](#)
- [‘i’ Conversion \(Signed Decimal Integer\)](#)
- [‘n’ Conversion \(Number of Characters Printed\)](#)
- [‘o’ Conversion \(Unsigned Octal Integer\)](#)
- [‘p’ Conversion \(Pointer\)](#)
- [‘s’ Conversion \(String\)](#)
- [‘u’ Conversion \(Unsigned Decimal Integer\)](#)
- [‘x’ or ‘X’ Conversion \(Unsigned Hexadecimal Integer\)](#)

The following sections provide details about the characteristics of each type of conversion.

“%” Conversion (Plain ‘%’)

A ‘%’ is printed. There is no argument.

‘c’ Conversion (Character As Integer)

The argument is an **int**, the least significant character of which is printed.

'd' Conversion (Signed Decimal Integer)

The argument is an **int** that prints as a signed decimal number.

Precision specifies the minimum number of digits to appear. If the value can be represented in fewer than Precision digits, it is padded with leading zeros. If Precision is 0 (zero) and the argument is 0 (zero), the converted value consists of no characters. If Precision is not specified, the default is 1 (one).

'e' or 'E' Conversion (Scientific Notation Double or Float)

The argument is a **double** (or a **float**) that prints in decimal notation. A **float** is converted to a **double** when passed as an argument. If the argument is negative, there is a leading minus sign.

If Precision is 0 (zero), no decimal point prints. If Precision is 0 (zero) and # appears, a decimal point prints. If Precision is not specified, the default is 6.

If the # flag appears, a decimal point is always printed, even if it is not followed by any digits.

The %e and %E conversions print the following, in the given order:

- one digit
- a decimal point
- Precision digits after the decimal point
- an 'e' (for %e) or an 'E' (for %E)
- the sign of the exponent
- the exponent (at least two digits)

'f' Conversion (Decimal Double Or Float)

The argument is a **double** (or **float**) that prints in decimal notation. A **float** is converted to a **double** when passed as an argument. If the argument is negative, there is a leading minus.

If Precision is 0 (zero), no decimal point prints. If Precision is 0 (zero) and # appears, a decimal point prints. If Precision is not specified, the default is 6. If the number has no integral portion and Precision is not 0 (zero), a '0' prints before the decimal point.

If the # flag appears, a decimal point is always printed, even if it is not followed by any digits.

The %f conversion prints the following, in the given order:

- the integral portion of the number
- a decimal point
- Precision digits after the decimal point

'g' or 'G' Conversion (Decimal Double or Float as 'e' or 'f')

The argument is a **double** (or **float**) that prints in style 'e', 'E', or 'f' (depending on the size of the exponent).

- If the exponent is less than -4 or greater than Precision, the %g conversion prints the argument in %e format, while the %G conversion prints it in %E format.
- If the exponent is greater than or equal to -4, or less than Precision, the %g and %G conversions print the argument in %f format.

A **float** is converted to a **double** when passed as an argument. If the argument is negative, there is a leading minus. By default, trailing zeros are removed from the result, and a decimal point appears only if it is followed by a digit.

Precision specifies the number of significant digits. If Precision is 0 (zero), the converted value consists of no characters, and no decimal point prints. If Precision is 0 (zero) and # appears, a decimal point prints. If Precision is not specified, the default is 6.

If the # flag appears, a decimal point is *always* printed (even if it is not followed by any digits), and trailing zeros are *not* removed. (This contrasts with the default behavior.)

'i' Conversion (Signed Decimal Integer)

The argument is an **int** that prints as a signed decimal number.

Precision specifies the minimum number of digits to appear. If the value can be represented in fewer than Precision digits, it is padded with leading zeros. If Precision is 0 (zero) and the argument is 0 (zero), the converted value consists of no characters. If Precision is not specified, the default is 1 (one).

'n' Conversion (Number of Characters Printed)

The argument is a pointer to an integer into which is written the number of characters printed thus far by the current call to **printf()**.

'o' Conversion (Unsigned Octal Integer)

The argument is an **int** that prints as an unsigned octal number.

Precision specifies the minimum number of digits to appear. If the value can be represented in fewer than Precision digits, it is padded with leading zeros. If Precision is 0 (zero) and the argument is 0 (zero), the converted value consists of no characters. If Precision is not specified, the default is 1 (one).

If the # flag appears and the result is not zero, '0' is prepended to the result.

'p' Conversion (Pointer)

The argument is a pointer to an object. The action taken is system- dependent — see the [System issues](#) section for more information.

's' Conversion (String)

The argument is a (**char** *) string.

The %s conversion prints characters from the string argument until either a NUL is encountered (which is not printed), or Precision characters have been printed. If Precision is not explicitly set, characters are printed until a NUL is found.

'u' Conversion (Unsigned Decimal Integer)

The argument is an **int** that prints as an unsigned decimal number.

Precision specifies the minimum number of digits to appear. If the value can be represented in fewer than Precision digits, it is padded with leading zeros. If Precision is 0 (zero) and the argument is 0 (zero), the converted value consists of no characters. If Precision is not specified, the default is 1 (one).

'x' or 'X' Conversion (Unsigned Hexadecimal Integer)

The argument is an **int** that prints as an unsigned hexadecimal number. The letters 'abcdef' print for %x conversions, while 'ABCDEF' print for %X conversions. By default, trailing zeros are removed from the result, and a decimal point appears only if it is followed by a digit.

Precision specifies the minimum number of digits to appear. If the value can be represented in fewer than Precision digits, it is padded with leading zeros. If Precision is 0 (zero) and the argument is 0 (zero), the converted value consists of no characters. If Precision is not specified, the default is 1 (one).

If the # flag appears, a decimal point is always printed (even if it is not followed by any digits), and trailing zeros are not removed; also, '0x' is prepended to the %x result and '0X' is prepended to the %X result.

Return Value

On success, **printf()** returns the number of characters printed. If a write error occurs, **printf()** returns a negative value.

System issues

On DOS and Windows NT, for a %p conversion specification, the argument is taken to be a pointer to an object. The value of the pointer (which is the address of the object) is printed in hexadecimal.

On 386 family architectures, if **sizeof(int)*2** is 4, the segment value is printed in four hex digits followed by a colon. The offset follows in **sizeof(int)*2** hex digits. On any other architectures, the value of the pointer is printed in **(sizeof(int))*2** hex digits.

On most UNIX systems, for a %p conversion specification, the argument is taken to be a pointer to an object. The value of the pointer is printed in hexadecimal.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

Example

See the file `printf.c` in the *install_dir/docs/examples/* directory on your distribution.

putc()

Write a character to a file

Targets

putc() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>          /* Required */
int putc(int c, FILE *stream);
```

Description

putc() is equivalent to [fputc\(\)](#).

NOTE **putc()** is provided both as a macro and as a function. When **putc()** is used as a macro, *stream* is evaluated more than once.

Return Value

On success, **putc()** returns the character *c*. If a write error occurs, **putc()** returns [EOF](#).

Surprises

putc() takes *c* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to **putc()**, it is automatically coerced to type **int**.

See Also

[I/O Functions](#) on page 427

typedef struct FILE — Information for controlling a stream

Example

See the file *getc.c* in the *install_dir/docs/examples/* directory on your distribution.

putch() and _putch()

Write a character to the screen

Targets

putch() and **_putch()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

putch() is Non-ANSI; **_putch()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <conio.h>
int putch(int c);
```

Description

putch() writes the character *c* directly to the screen.

Return Value

putch() returns the character *c*.

Surprises

putch() takes *c* as an **int** rather than as a **char** argument. If an object of type **char** is passed to **putch()**, it is automatically coerced to type **int**.

See Also

[I/O Functions](#) on page 427

Example

See the file *putch.c* in the *install_dir/docs/examples/* directory on your distribution.

putchar()

Write a character to standard output

Targets

putchar() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required for macro */
int putchar(int c);
```

Description

putchar() is equivalent to **fputc(c, stdout)**.

Note **putchar()** is provided both as a macro and as a function. The macro **putchar(c)** expands to **putc(c, stdout)**. **putc()** itself might also be implemented as a macro.

Return Value

On success, **putchar()** returns the character *c*. If a write error occurs, **putchar()** returns EOF.

Surprises

putchar() takes *c* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to **putchar()**, it is automatically coerced to type **int**.

See Also

[**fputc\(\)** — Write a character to a file](#)

[I/O Functions](#) on page 427

Example

See the file *putchar.c* in the *install_dir/docs/examples/* directory on your distribution.

putenv() and _putenv()

Add or modify an environment variable

Targets

putenv() and **_putenv()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	x	x	x

ANSI

putenv() is Non-ANSI; **_putenv()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdlib.h> /* Required */
int putenv(char * "envstring");
int _putenv(char * "envstring");
```

Description

putenv() and **_putenv()** take an argument *envstring* of the form *varname*=*string* and search the environment block of the currently running process for an entry corresponding to *varname*.

If the functions find *varname* in the environment block, they replace the existing value of *varname* with the new value *string*. If the functions do not find *varname* in the environment block, they add *varname*=*string* to the environment.

These are valid uses of **putenv()** and **_putenv()**:

```
_putenv("HOME=/home/me");
...
char str[80];
strcpy(str, "HOME=/home/you");
putenv(str);
```

You can set an environment variable to an empty value by specifying an empty *string*, as in **putenv("HOME")**.

Environment Block

Each program inherits from its parent process a data structure called an *environment block*. The environment block is a series of null-terminated strings, of the form *name*=*parameter*, that define the environment in which a process executes.

For example, on DOS, this environment variable setting is used by the MetaWare C compiler to find include files during compilation:

```
IPATH=C:\HIGHC\INC\
```

NOTE The environment block of a parent process is not modified when the environment block of a child process is changed. When a program terminates, the environment reverts back to the original environment block of the parent process.

System issues

On UNIX, this function can be provided by the system library.

Return Value

If successful, **putenv()** and **_putenv()** return 0 (zero). If not successful, the functions return -1.

See Also

[getenv\(\)](#) — Get the value of an environment variable

Example

See the file *putenv.c* in the *install_dir/docs/examples/* directory on your distribution.

puts()

Write a string to standard output

Targets

puts() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>
int puts(const char *s);
```

Description

puts() writes the string *s* to the stream associated with `stdout`, and appends a newline. The terminating NUL character ('\0') is not written.

Return Value

puts() returns 0 (zero) if the operation is successful. If the operation fails, **puts()** sets global variable `errno` such that a call to **perror()** will result in an error message describing the reason for the failure, and returns EOF.

Surprises

puts() and **fputs()** are inconsistent: **fputs()** writes exactly the string *s*, while **puts()** writes *s* followed by a newline.

See Also

[fputs\(\)](#) — Write a string to a file

[I/O Functions](#) on page 427

[perror\(\)](#) — Print an error message

Constant [EOF](#) — End-of-file indicator

Global variable [errno](#) — An int used to record error numbers

Example

See the file `gets.c` in the *install_dir/docs/examples/* directory on your distribution.

putw() and _putw()

Write an integer to a file

Targets

putw() and **_putw()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

putw() is Non-ANSI; **_putw()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>          /* Required */
int putw(int int1, FILE *stream);
```

Description

putw() writes *int1* to the current position of *stream*.

Return Value

putw() returns the integer written to *stream*.

See Also

[I/O Functions](#) on page 427

typedef struct FILE — Information for controlling a stream

Example

See the file *putw.c* in the *install_dir/docs/examples/* directory on your distribution.

qsort()

Perform a quicksort

Targets

qsort() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>          /* Required */
void qsort(void *base, size_t nmemb, size_t size,
           int (*cmp)(const void *ptr1, const void *ptr2));
```

Description

qsort() sorts the array of *nmemb* objects of size *size*, pointed to by *base*. The array is sorted into ascending order according to a user-supplied comparison function *cmp()*.

Function *cmp()* is called with two pointers to array members (*ptr1* and *ptr2*) and must return an integer less than, equal to, or greater than 0 (zero) depending on whether the first array member is less than, equal to, or greater than the second.

- If **ptr1* is less than **ptr2*, the *cmp()* function returns an **int** less than 0 (zero).
- If **ptr1* is greater than **ptr2*, the *cmp()* function returns an **int** greater than 0 (zero).
- If **ptr1* is equal to **ptr2*, the *cmp()* function returns 0 (zero).

See Also

[bsearch\(\)](#) — Perform a binary search

[**Ifind\(\) and _Ifind\(\)**](#) — Search a linear list for a matching value, without modifying the list

[**Isearch\(\) and _Isrch\(\)**](#) — Search a linear list for a matching value, modifying if necessary

Example

See the file `qsort.c` in the *install_dir/docs/examples/* directory on your distribution.

raise()

Raise a signal

Targets

`raise()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <signal.h>
int raise(int sig);
```

Description

`raise()` sends the signal *sig* to the executing program. If a signal handler has been defined for *sig*, `raise()` will cause its execution; otherwise, the default action will be taken.

For a list of the available signals and their default actions, see the [SIG*](#) constants and the macros [**SIG_DFL**](#) and [**SIG_ERR**](#) and [**SIG_IGN**](#), [**SIG_DFL**](#) and [**SIG_ERR**](#) and [**SIG_IGN**](#), and [**SIG_DFL**](#) and [**SIG_ERR**](#) and [**SIG_IGN**](#) in [Constants and Globals](#) on page 319. These constants and macros are defined in `signal.h`.

Note If a call is made to `signal()`, `raise()`, or `_kill()`, the library module `signal.obj` is linked into the executable, providing a set of interrupt handlers markedly different from those supplied by the operating system.

If a call to one of these functions is never made in a program, the routines used to handle interrupts will be determined by the operating system. Keep this in mind when using interrupt routines in a program that calls one of these functions.

Return Value

`raise()` returns 0 (zero) if successful. If not successful, it returns a non-zero value.

See Also

[SIG*](#) — [Arguments to signal\(\) and raise\(\)](#)

[**SIG_DFL**](#) and [**SIG_ERR**](#) and [**SIG_IGN**](#), [**SIG_DFL**](#) and [**SIG_ERR**](#) and [**SIG_IGN**](#), and [**SIG_DFL**](#) and [**SIG_ERR**](#) and [**SIG_IGN**](#) — Macros; arguments to `signal()`

[signal\(\)](#) — [Set up a signal handler](#)

rand()

Generate a pseudo-random number

Targets

rand() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>
int rand(void);
```

Description

rand() generates a pseudo-random integer. The sequence computed by rand() has a period of 2^{32} . Use srand() to seed the pseudo-random number generator.

Return Value

rand() returns the pseudo-random integer that it generates.

See Also

[srand\(\)](#) — Seed the pseudo-random number generator

Example

See the file `rand.c` in the `install_dir/docs/examples/` directory on your distribution.

read() and _read()

Read data from a file using a handle

Targets

read() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

_read() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

read() is Non-ANSI; _read() is Extra-ANSI.

Reentrancy

Both the functions are Not-Reentrant

Synopsis

```
/* Required for DOS and Windows targets: */
#include <io.h>

/* Required for all other targets: */
#include <unistd.h>

int read(int handle, char *buf, unsigned int count);
int _read(int handle, char *buf, unsigned int count);
```

Description

read() and **_read()** read *count* bytes from the file currently opened as *handle*, into the buffer starting at *buf*. The read occurs at the current position in the file, and the current position is changed to point to the next character in the file after the data that was read into the buffer.

A text file has three characters that receive special treatment: CTRL-Z, carriage-return (CR), and line-feed (LF).

- If a file is opened using [**O_TEXT**](#), a CTRL-Z character is considered a logical end-of-file, and the read will not proceed beyond that point.
- Carriage-return/line-feed (CR/LF) characters terminate each line of text. However, C uses only a line-feed character, so the carriage-return character is removed from the input data in this situation.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

read() and **_read()** return the actual number of bytes placed in the buffer. The return value might be less than *count*, if there were fewer than *count* bytes remaining in the file, or if this is a text file and there were CR/LF pairs from which the carriage return was deleted. **read()** and **_read()** return 0 (zero) to indicate end-of-file.

On failure, **read()** and **_read()** return -1 and set **errno** to [**EBADF** \(“Bad” \(invalid or inaccessible\) file handle\)](#).

See Also

[I/O Functions](#) on page 427

[File-Related Functions](#) on page 429

Example

See the file *read.c* in the *install_dir/docs/examples/* directory on your distribution.

_readdir()

Read a POSIX directory stream

Targets

_readdir() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <dirent.h>          /* Required */
struct _dirent *_readdir(_DIR *dir)
```

Description

`_readdir()` reads the current directory entry in the directory stream pointed to by `dir`. Subsequent calls to `_readdir()` on the same stream return subsequent directory entries, including system files, hidden files, the entries ‘.’ and ‘..’, and other subdirectories.

After a successful call to `_readdir()`, the function returns a pointer to a [`_dirent`](#) structure. This structure corresponds to a single directory entry and is overwritten by successive calls to `_readdir()` using the same directory stream.

The [`_dirent`](#) structure, defined in `dirent.h`, contains the field `d_name`, which contains the file name for the directory entry. `d_name` is defined as follows:

```
char d_name[];
```

If directory entries are created or deleted while the directory stream is being read, `_readdir()` might not reflect the change. Use `_rewinddir()` to reset the stream (or `_opendir()` to reopen the stream) to ensure that `_readdir()` reflects the current state of the directory.

Return Value

If successful, `_readdir()` returns a pointer to a directory entry in the stream. If `dir` does not point to a valid directory stream, or if `_readdir()` attempts to read past the end of the stream, `_readdir()` returns [`NULL`](#).

See Also

[`closedir\(\)`](#) — [Close a POSIX directory stream](#)
[`opendir\(\)`](#) — [Open a POSIX directory stream](#)
[`rewinddir\(\)`](#) — [Reset a POSIX directory stream](#)
[struct `_dirent`](#) — [A POSIX file name for `_readdir\(\)`](#)

Example

See the file `opendir.c` in the `install_dir/docs/examples/` directory on your distribution.

realloc()

Reallocate storage allocated by `calloc()` or `malloc()`

Targets

`realloc()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>          /* Required */
void *realloc(void *ptr, size_t size);
```

Description

realloc() changes the size of the object pointed to by *ptr*, making it *size* bytes. The contents of the object are unchanged up to the lesser of *size* and the original size of the object. If the space cannot be allocated, **realloc()** returns **NULL**, and the original object is unchanged. Newly allocated storage (if any) is uninitialized.

If *ptr* is **NULL**, **realloc()** behaves like the **malloc()** function. If *ptr* was not previously returned by **malloc()**, **calloc()**, or **realloc()**, or if the space was previously freed by a call to **free()** or **realloc()**, the behavior is undefined. If *size* is 0 (zero), the space is freed.

Return Value

If successful, **realloc()** returns a pointer to the newly allocated, possibly moved, object. If not successful, **realloc()** returns [NULL](#) and the object is unchanged.

See Also

[calloc\(\)](#) — [Dynamically allocate zero-initialized storage for objects](#)

[free\(\)](#) — [Release storage allocated by calloc\(\), malloc\(\), or realloc\(\)](#)

[malloc\(\)](#) — [Dynamically allocate uninitialized storage](#)

Example

See the file `realloc.c` in the *install_dir/docs/examples/* directory on your distribution.

remove()

Delete a file

Targets

remove() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>
int remove(const char *pathname);
```

Description

remove() removes the file named by the string *pathname*.

CAUTION If **remove()** is called on an open file, the behavior is implementation-defined.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

If successful, **remove()** returns 0 (zero). If not successful, **remove()** sets global variable **errno** such that a call to **perror()** will result in an error message describing the reason for the failure, and returns the (non-zero) value of **errno**.

See Also

[fclose\(\)](#) — Close a file

[perror\(\)](#) — Print an error message

[rename\(\)](#) — Change the name of a file

Global variable [errno](#) — An int used to record error numbers

Example

See the file `remove.c` in the *install_dir/docs/examples/* directory on your distribution.

rename()

Change the name of a file

Targets

rename() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>
int rename(const char *old, const char *new);
```

Description

rename() changes the name of the file named by the string *old* to the name contained in the string *new*. There will no longer be a file whose name is contained in the string *old*.

CAUTION If **rename()** is called on an open file, the behavior is implementation-defined.

Return Value

rename() returns 0 (zero) if the operation is successful. If the operation fails, **rename()** sets global variable **errno** such that a call to **perror()** will result in an error message describing the reason for the failure, and returns the (non-zero) value of **errno**.

System issues

On UNIX, **rename()** is not supported on the Sun3 or the Sun4.

See Also [perror\(\) — Print an error message](#)[File-Related Functions](#) on page 429Global variable [errno — An int used to record error numbers](#)**Example**See the file `rename.c` in the `install_dir/docs/examples/` directory on your distribution.

rewind()

Seek to the beginning of a file

Targets

rewind() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>          /* Required */
void rewind(FILE *stream);
```

Description

rewind() is equivalent to the following sequence of calls, except that rewint() has no return value:

```
fseek(stream, 0, SEEK_SET);
clearerr(stream);
```

See Also[clearerr\(\) — Clear end-of-file and error flags](#)[fseek\(\) — Move file pointer to new location in file](#)[ftell\(\) — Determine current value of a file pointer](#)[SEEK_SET — Macros used to specify position in file](#)[typedef struct FILE — Information for controlling a stream](#)[File-Related Functions](#) on page 429**Example**See the file `rewind.c` in the `install_dir/docs/examples/` directory on your distribution.

_rewinddir()

Reset a POSIX directory stream

Targets

_rewinddir() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <dirent.h>           /* Required */  
int _rewinddir(_DIR *dir)
```

Description

_rewinddir() resets a directory stream so the next call to _readdir() will read the first directory entry. It also ensures that the directory stream accounts for any directory entries that might have been created or deleted since the last call to [_opendir\(\)](#) or [_rewinddir\(\)](#).

Return Value

_rewinddir() returns 0 (zero) if successful. If not successful, it returns a non-zero value.

See Also

[closedir\(\)](#) — [Close a POSIX directory stream](#)

[opendir\(\)](#) — [Open a POSIX directory stream](#)

[readdir\(\)](#) — [Read a POSIX directory stream](#)

struct [_dirent](#) — [A POSIX file name for readdir\(\)](#)

Example

See the file `opendir.c` in the `install_dir/docs/examples/` directory on your distribution.

rmdir() and _rmdir()

Remove a directory

Targets

rmdir() works on the following target platforms:

EMB	UNIX	DOS	NT
—	x	x	x

_rmdir() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

rmdir() is Non-ANSI; _rmdir() is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
/* Required for DOS and Windows targets: */
#include <direct.h>

/* Required for all other targets: */
#include <unistd.h>

int rmdir (char *path);
int _rmdir (char *path);
```

Description

rmdir() and **_rmdir()** remove the directory specified by *path*. These functions result in an error if the directory is not empty, or if *path* refers to the current working directory or the root directory.

System issues

On UNIX, this function can be provided by the system library.

Return Value

rmdir() and **_rmdir()** return 0 (zero) if the directory was deleted. If the call results in an error, **rmdir()** and **_rmdir()** return -1 and set the global variable `errno` to one of the following values:

EACCES — [Access error; permission denied](#)

ENOENT — [File or directory not found; pathname element does not exist](#)

An error that sets `errno` to **EACCESS** can mean one of the following:

- *path* refers to the current directory.
- *path* refers to the root directory.
- The directory specified by *path* is not empty.
- *path* does not refer to a directory (perhaps a file of the same name exists).

See Also

[chdir\(\) and _chdir\(\)](#) — Change the current working directory

[getcwd\(\) and _getcwd\(\)](#) — Get full pathname of the current working directory

Example

See the file `rmdir.c` in the *install_dir/docs/examples/* directory on your distribution.

_rmemcpy()

Copy from one place in memory to another

Targets

_rmemcpy() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */  
void *_rmemcpy(void *dest, const void *source, size_t n);
```

Description

`_rmemcpy()` copies *n* bytes from the location referenced by *source* to the location referenced by *dest*. `_rmemcpy()` copies from right to left (high addresses to low addresses).

If *dest* < *source* < *dest* + *n* (that is, if the source and destination areas overlap, with the source area beginning within the destination area) use `memcpy()`.

CAUTION `_rmemcpy()` writes on *n* bytes.

Return Value

`_rmemcpy()` returns a pointer to *dest*.

Surprises

`_rmemcpy()` copies the second argument to the first.

See Also

[`memcpy\(\)` — Copy from one place in memory to another](#)

[`memset\(\)` — Duplicate a character across an area of memory](#)

[`rstrncpy\(\)` — Copy a number of characters from one string into another](#)

[`strcpy\(\)` — Copy the characters of one string into another](#)

[`strncpy\(\)` — Copy a number of characters from one string into another](#)

Example

See the file `rmemcpy.c` in the *install_dir/docs/examples/* directory on your distribution.

_rotl() and _rotr()

Rotate an **int** left or right

Targets

The `_rot*`() functions work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Both functions are Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h>  
unsigned int _rotl(unsigned int value, int bits);  
unsigned int _rotr(unsigned int value, int bits);
```

Description

_rotl() rotates *value* to the left by the number of bits given in *bits*.

_rotr() rotates *value* to the right by *bits* bits.

Return Value

_rotl() returns *value* rotated to the left.

_rotr() returns *value* rotated to the right.

See Also

[_crotl\(\) and _crotr\(\)](#), [_crotl\(\) and _crotr\(\)](#) — [Rotate a char left or right](#)

[_lrotl\(\) and _lrotr\(\)](#), [_lrotl\(\) and _lrotr\(\)](#) — [Rotate a long int left or right](#)

Example

See the file *rot.c* in the *install_dir/docs/examples/* directory on your distribution.

_rstrcpy()

Copy one string onto another

Targets

_rstrcpy() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */
char *_rstrcpy(char *dest, const char *source);
```

Description

_rstrcpy() copies the string *source* to the string *dest*, including the NUL that terminates *source*.

_rstrcpy() copies from right to left (high addresses to low addresses).

If *dest* < *source* < *dest* + n (that is, if the source and destination areas overlap, with the source area beginning within the destination area) use **strcpy()**.

CAUTION _rstrcpy() writes until the end of string *source* is encountered, without regard for the length of *dest*.

Return Value

_rstrcpy() returns a pointer to *dest*.

Surprises

_rstrcpy() copies the second argument to the first.

See Also

[_memcpy\(\)](#) — Copy from one place in memory to another
[_memset\(\)](#) — Duplicate a character across an area of memory
[_rmemcpv\(\)](#) — Copy from one place in memory to another
[_rstrncpy\(\)](#) — Copy a number of characters from one string into another
[_strcpy\(\)](#) — Copy the characters of one string into another
[_strncpy\(\)](#) — Copy a number of characters from one string into another

Example

See the file `rstrcpy.c` in the `install_dir/docs/examples/` directory on your distribution.

_rstrncpy()

Copy a number of characters from one string into another

Targets

`_rstrncpy()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */  
char *_rstrncpy(char *dest, const char *source, size_t n);
```

Description

`_rstrncpy()` copies exactly *n* characters from the string *source* into *dest*, from right to left (high addresses to low addresses).

- If a NUL terminates *source* before *n* characters have been copied, *dest* is NUL padded.
- If no NUL is encountered before *n* characters have been written into *dest*, the resulting character array is not NUL terminated.

If *dest* < *source* < *dest* + *n* (that is, if the source and destination areas overlap, with the source area beginning within the destination area) use `strncpy()`.

CAUTION `_rstrncpy()` writes on *n* bytes.

After a call to `_rstrncpy()`, it might not be safe to pass *dest* to functions that expect a string, because *dest* might not be terminated with a NUL.

Return Value

`_rstrncpy()` returns a pointer to *dest*.

Surprises

`_rstrncpy()` copies the second argument to the first.

See Also

[memcpy\(\)](#) — Copy from one place in memory to another

[memset\(\)](#) — Duplicate a character across an area of memory

[_rmemcpy\(\)](#) — Copy from one place in memory to another

[_rstrcpy\(\)](#) — Copy one string onto another

[strcpy\(\)](#) — Copy the characters of one string into another

[strncpy\(\)](#) — Copy a number of characters from one string into another

Example

See the file `rstrncpy.c` in the *install_dir/docs/examples/* directory on your distribution.

scanf()

Read values from `stdin`

Targets

`scanf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

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

Description

`scanf()` reads values from standard input. `scanf()` interprets each argument that follows *format* as a pointer to the type specified by the corresponding conversion specification.

If an argument is not the expected pointer, it is still treated as if it were, potentially causing some arbitrary place in memory to be overwritten with the result of the conversion.

If the incorrect argument is not the same size as a pointer, subsequent arguments might be misinterpreted as well. In most systems, the pointer is expected to be a hexadecimal number.

CAUTION If fewer arguments are passed to `scanf()` than *format* specifies, arbitrary places in memory are overwritten with the results of the specified conversions.

The discussion of the format string and conversion specifications in this section applies to `*scanf()`, all the functions in the `scanf()` family:

- [cscanf\(\) and _cscanf\(\)](#) — Read from standard input
- [fscanf\(\)](#) — Read values from a file
- [scanf\(\)](#) — Read values from `stdin`
- [sscanf\(\)](#) — Read values from a string
- [vfscanf\(\) and _vfscanf\(\)](#) — Read values from a file using var-arg macros
- [vscanf\(\) and _vscanf\(\)](#) — Read values from `stdin` using var-arg macros
- [vsscanf\(\) and _vsscanf\(\)](#) — Read values from a string using var-arg macros

Format string

format is an arbitrary sequence of three types of elements: conversion specifications, whitespace characters, and non-whitespace characters.

- A conversion specification in *format* causes characters in the input stream to be read and converted to a specified type, as described in the following sections. In most cases there should be a corresponding argument *A* that is a pointer to an object of that type.
- Any whitespace character in *format* that is not a part of a conversion specification causes input to be consumed up to the next non-whitespace character.
- Any non-whitespace character in *format* that is not a part of a conversion specification must match the next character in the input stream. If such a character does not match the next input character, `scanf()` returns.

The correspondence between conversion specifications and arguments is simple: the first conversion specification that expects an argument corresponds to the first argument *A* following *format*, the second specification corresponds to the second argument, and so on.

CAUTION Too few arguments:

If the number of actual arguments is less than the number of arguments specified in *format*, `scanf()` wanders merrily through memory, assuming that whatever it finds is a pointer to the type of object required by the specification in *format*, and assigning into that object.

Too many arguments:

If the number of arguments is greater than the number specified by *format*, the excess arguments are evaluated by the standard function-call mechanism, but are otherwise ignored.

Conversion Specifications

This is the form of a conversion specification:

`'%' '*'? Field_width? Size? C_Char`

The components of the conversion specification have the following meanings:

<u>Field_width</u>	-> Digits;
<u>Size</u>	->'h' 'l' 'L';
<u>C_Char</u>	->'d' 'i' 'o' 'u' 'x' 'X' 'f' 'e' 'E' 'g' 'G' 'c' 's' 'p' 'n' '%' '['

(See [Regular Expressions](#) on page 18 for an explanation of this notation.)

- '%' The '%' character sets off a conversion specification from ordinary characters.
- '*' The '*' character causes the next value in the input stream to be converted, but suppresses the assignment of the resulting value. All the computation (and concomitant reading of input) is done, but the result is ignored. No argument corresponds to a conversion specification containing '*'.

The input value to be converted generally extends from the next input character to the first character in the input stream that cannot be a part of that value. Trailing whitespace is left unread unless matched in the format string.

Except for the `c` and `[` conversions, leading whitespace is skipped. For example, if `format` specifies that a decimal integer is to be found in the input stream, first all whitespace is skipped, then all characters that can be part of a decimal integer (all `Digits`) are read.

The conversion ends with the first character that cannot be part of a decimal integer (including whitespace characters), and that conflicting input character remains unread, available as the first character of the next input value.

It is possible for an input value to consist of zero characters for some conversions. For instance, when a decimal integer is expected and the next character in the input is the character 'k', no value is assigned, the conflicting character remains in the input stream, and `scanf()` returns.

If a conversion specification is invalid, that is, if the conversion character is not one of those listed, global variable `errno` is set and that conversion is ignored — no input is read and no argument is assigned into.

The Field_width Specification

`Field_width` can be used to impose a maximum on the number of characters that make up an input value.

The Size Specification

`Size` specifies the size of the receiving object for an argument *A* that points to a type that comes in more than one size; for example, `int` versus `long int`.

`Size` is mentioned in the following [C Char](#) component definitions for each conversion it affects. If `Size` is specified for a conversion that it cannot affect, it is ignored.

Size of 'd', 'i', 'o', 'x', and 'X' Conversions:

The corresponding argument *A* should be a pointer to `int` if `Size` is not specified.

- If `Size` is `h`, *A* is a pointer to `short int`.
- If `Size` is `l`, *A* is a pointer to `long int`.
- If `Size` is `L` or `ll`, *A* is a pointer to `long long`.

Size of 'u' Conversions:

The corresponding argument *A* should be a pointer to `unsigned int` if `Size` is not specified.

- If `Size` is `h`, *A* is a pointer to `unsigned short int`.
- If `Size` is `l`, *A* is a pointer to `unsigned long int`.
- If `Size` is `L` or `ll`, *A* is a pointer to `unsigned long long`.

Size of e, E, f, g, and G Conversions:

The corresponding argument *A* should be a pointer to **float** if **Size** is not specified.

- If **Size** is 1, *A* is a pointer to **double**.
- If **Size** is L or lL, *A* is a pointer to **long double**.

The C_Char Specification

C_Char specifies the type of value expected in the input stream (and thus it implicitly specifies the type of the object pointed to by the corresponding argument *A*). Each **C_Char** in a format specification is modified by the accompanying **Field_width** and **Size** components.

The following conversion characters are defined here:

- [‘c’ Conversion \(One Character\)](#)
- [‘d’ Conversion \(Decimal Integer\)](#)
- [‘e’, ‘E’, ‘f’, ‘g’, and ‘G’ Conversions \(Floating-Point Numbers\)](#)
- [‘i’ Conversion \(Decimal, Hexadecimal, or Octal Integer\)](#)
- [‘n’ Conversion \(Number of Characters Read\)](#)
- [‘o’ Conversion \(Octal Integer\)](#)
- [‘p’ Conversion \(Pointer\)](#):
- [‘s’ Conversion \(Character String\)](#)
- [‘u’ Conversion \(Unsigned Decimal Integer\)](#)
- [‘x’ and ‘X’ Conversions \(Hexadecimal Integers\)](#)
- [‘%’ Conversion \(Plain ‘%’\)](#)
- [‘\[’ Conversion \(Begin Sequence Description\)](#)

For an explanation of the notation used in these descriptions, see [Regular Expressions](#) on page 18.

‘c’ Conversion (One Character)

One character is expected. The corresponding argument *A* should be a pointer to **char**. Whitespace is not skipped in this case (to read the next non-whitespace character, use “%1s”).

Size has no effect on argument *A*. If **Field_width** is given, **Field_width** characters are read; *A* should point to a character array large enough to hold them. No NUL character ('\0') is appended.

‘d’ Conversion (Decimal Integer)

A decimal integer of this form is expected:

(‘+’ | ‘-’)? Digits

The corresponding argument *A* should be a pointer to **int** if **Size** is not specified. (See the preceding section, [The Size Specification](#), for information about the effect of **Size** on argument *A*.)

‘e’, ‘E’, ‘f’, ‘g’, and ‘G’ Conversions (Floating-Point Numbers)

A floating-point number of this form is expected:

Sign? ((Digits (‘.’) Digits) | (‘.’ Digits))
((‘E’ | ‘e’) Sign? Digits?)?

The corresponding argument *A* should be a pointer to **float** if **Size** is not specified. (See the preceding section, [The Size Specification](#), for information about the effect of **Size** on argument *A*).

If a conflicting character appears in the input before a floating-point value is found, no value is assigned and the conflicting character remains in the input stream.

'i' Conversion (Decimal, Hexadecimal, or Octal Integer)

An integer of the following form is expected:

`('+' | '-')? ('0' ('x' | 'X')?)? Hexdigits`

The corresponding argument *A* should be a pointer to **int** if **Size** is not specified. (See the preceding section, [The Size Specification](#), for information about the effect of **Size** on argument *A*.)

Following the optional sign:

- If the input begins with prefix 0x or 0X, the value is assumed to be a hexadecimal number.
- If the input begins with prefix 0 (but no x or X), the value is assumed to be an octal number.
- Otherwise, the value is assumed to be a decimal number.

Following the optional sign and the optional prefix, if the next character in the input is not a digit of the appropriate base, no value is assigned and the conflicting character remains in the input stream.

'n' Conversion (Number of Characters Read)

The argument *A* is a pointer to **int** into which is written the number of characters thus far read by the current call to **scanf()**. No input is consumed. [Size](#) has no effect on argument *A*.

'o' Conversion (Octal Integer)

An octal integer of the form **Odigits** is expected:

The corresponding argument *A* should be a pointer to **int** if **Size** is not specified. (See the preceding section, [The Size Specification](#), for information about the effect of **Size** on argument *A*.)

If the next character in the input is not an octal digit, no value is assigned and the conflicting character remains in the input stream.

'p' Conversion (Pointer):

A pointer is expected. The corresponding argument *A* should be a pointer to a pointer to **void**. See the [System issues](#) section that follows.

's' Conversion (Character String)

A character string is expected. The corresponding argument *A* should be a pointer to a character array large enough to accept the string and a NUL character ('\0') that is automatically appended to the result. The string is terminated in the input by any whitespace character. [Size](#) has no effect on argument *A*.

We advise using [Field_width](#) with this specification; otherwise, the array might be overrun for some input. [Field_width](#) does not include the automatically appended NUL character.

'u' Conversion (Unsigned Decimal Integer)

An unsigned decimal integer of the form **Digits** is expected.

The corresponding argument *A* should be a pointer to **unsigned int** if **Size** is not specified. (See the preceding section, [The Size Specification](#), for information about the effect of **Size** on argument *A*.)

If the next character in the input is not a decimal digit, no value is assigned and the conflicting character remains in the input stream.

'x' and 'X' Conversions (Hexadecimal Integers)

A hexadecimal integer of the form **Hexdigits** is expected.

The corresponding argument *A* should be a pointer to **int** if **Size** is not specified. (See the preceding section, [The Size Specification](#), for information about the effect of **Size** on argument *A*.)

If the next character in the input is not a hexadecimal digit, no value is assigned and the conflicting character remains in the input stream.

'%' Conversion (Plain '%')

A % is expected. No assignment occurs. [Size](#) has no effect on argument *A*.

'[Conversion (Begin Sequence Description)

A string that is not delimited by whitespace is expected. The corresponding argument *A* should be a pointer to a character array. [Size](#) has no effect on argument *A*.

The left bracket is followed by a sequence of characters, then a right bracket.

- If the first character in the sequence is not a caret (^), input characters are assigned to the array pointed to by *A* as long as they are in the sequence.
- If the first character is a caret, characters are assigned until a character from the sequence is encountered.

A NUL character ('\0') is appended to the string. We advise using [Field width](#) with this specification; otherwise the array might be overrun for some input. If the first character in the sequence is a right bracket (']'), **scanf()** returns only a string that begins with a right bracket.

NOTE A range of characters is indicated by using a minus sign ('-'), between characters. Using a range of characters is a non-ANSI feature.

Return Value

scanf() returns the number of input values that were successfully converted and assigned, or **EOF** if end-of-file is encountered before the first conflict or conversion. If no conflict occurs, **scanf()** returns when the end of the format string is encountered.

System issues

For a %p conversion, a pointer is expected. The corresponding argument *A* should be a pointer to a pointer to **void**.

On 386 family architectures, four hexadecimal digits are read; these are taken to be the segment value. Then a colon is read, then **sizeof(int)*2** hex digits. These hex digits are taken to be the offset.

On other architectures, no segment value or colon is read. The address read is assigned into the pointer pointed to by *A*. This conversion is guaranteed to work only on a value that is output from one of the [printf\(\)](#) function's conversions of a pointer (also %p) during the same run of a program.

If the stream is a text stream, DOS line terminators (\r\n) are converted into C terminators (\n). In addition, a CTRL-Z (\032) is interpreted as an end-of-file indicator and **EOF** is returned.

If the stream is a binary stream, no such conversion occurs, and CTRL-Z characters encountered in the input stream are not interpreted but returned as is.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

Example

See the file `scanf.c` in the `install_dir/docs/examples/` directory on your distribution.

_searchenv()

Search for a file using an environment variable

Targets

_searchenv() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */  
void _searchenv(const char *filename, const char *envvar, char *path);
```

Description

_searchenv() searches for a file in the domain specified by *envvar*.

- *filename* is the file to search for.
- *envvar* can be any environment variable that specifies directory paths, such as PATH, INCLUDE, LIB, IPATH, or any other user-defined variable.
- *path* points to a buffer in which **_searchenv()** stores the full pathname of the found file. If *filename* is not found, the buffer pointed to by *path* contains an empty null-terminated string.

_searchenv() first looks in the current working directory for *filename*.

- If *filename* is found, the full pathname of the file is copied into the buffer pointed to by *path*, which must have sufficient space to contain it.
- If *filename* is not found, **_searchenv()** searches the directories specified by environment variable *envvar*.

See Also

[getenv\(\)](#) — Get the value of an environment variable

[putenv\(\) and _putenv\(\)](#) — Add or modify an environment variable

[searchstr\(\)](#) — Search for a file in a list of directories

Example

See the file `searchenv.c` in the *install_dir/docs/examples/* directory on your distribution.

_searchstr()

Search for a file in a list of directories

Targets

`_searchstr()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */
void _searchstr(const char *file, const char *ipath, char *buf);
```

Description

`_searchstr()` searches for a file in the domain specified by *ipath*.

- *file* is the file to search for.
- *ipath* is a string containing pathname components in the same format as environment variables, such as PATH.
- *buf* points to a buffer in which `_searchstr()` stores the full pathname of the found file. If *file* is not found, the buffer pointed to by *buf* will contain an empty null-terminated string.

`_searchstr()` first looks in the current working directory for *file*.

- If *file* is not found, `_searchstr()` searches the directories specified by the variable *ipath*.
- If *file* is found, the full pathname of the file is copied into the buffer pointed to by *buf*, which must have sufficient space to contain it.

See Also

[getenv\(\)](#) — Get the value of an environment variable

[putenv\(\) and _putenv\(\)](#) — Add or modify an environment variable

[searchenv\(\)](#) — Search for a file using an environment variable

Example

See the file `searchst.c` in the *install_dir/docs/examples/* directory on your distribution.

setbuf()

Specify a buffer for a stream

Targets

`setbuf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
void setbuf(FILE *stream, char *buf);
```

Description

setbuf() causes a specified area of memory, instead of an automatically allocated buffer, to be used by the stream *stream*.

- *buf* references the beginning location of the memory area to be used. The memory area is [BUFSIZ](#) bytes in length. If *buf* is **NULL**, I/O on the stream is unbuffered.
- *stream* is the stream that will use the memory area.

A call to **setbuf(stream, buf)** is equivalent to a call to **setvbuf(stream, buf, [IOFBF](#) and [IOLBF](#) and [IONBF](#), [BUFSIZ](#))**.

If I/O has been done on the stream at the time **setbuf()** is called, the behavior is undefined.

CAUTION Deallocating the buffer that *buf* references, before the file is closed, can wreak havoc.

If *buf* points to local storage, **fclose()** should be explicitly called on *stream* before exiting the block containing the declaration of the buffer. Normal program termination involves closing all open files, which in turn involves flushing the buffers of all buffered files that are open for writing or update.

If the buffer referenced by *buf* is local and the block in which it is declared is exited before **fclose()** is called on *stream*, the buffer might be overwritten before it is flushed.

See Also

[File-Related Functions](#) on page 429

[typedef struct FILE — Information for controlling a stream](#)

setjmp()

Save a reference to the current calling environment for a subsequent non-local jump

Targets

setjmp() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <setjmp.h> /* Required */
int setjmp(jmp_buf env);
```

Description

setjmp() saves into *env* the information required for a return to the current calling environment, for later use by **longjmp()**.

CAUTION Correctly paired, **setjmp()** and **longjmp()** are fairly innocuous. If they are incorrectly paired, the behavior is undefined and can result in jumping to a non-existent environment.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return Value

setjmp() returns 0 (zero).

Surprises

A call to **setjmp()** cannot be used as an argument to a function call.

See Also

[longjmp\(\) — Execute a non-local jump](#)

Example

See the file `longjmp.c` in the `install_dir/docs/examples/` directory on your distribution.

setlocale()

Set the current locale

Targets

setlocale() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <locale.h>          /* Required */  
char *setlocale(int category, const char *locale);
```

Description

setlocale() changes or queries the current locale or portions thereof.

If *category* is **LC_ALL**, the entire locale is designated. The following constant values for *category* indicate other specific portions of the locale:

- [LC_COLLATE — Affects the strcoll\(\) and strxfrm\(\) functions.](#)
- [LC_CTYPE — Affects character and multibyte functions.](#)
- [LC_MONETARY — Affects monetary numeric formatting.](#)

- [LC_NUMERIC](#) — Affects non-monetary numeric formatting.
- [LC_TIME](#) — Affects the `strftime()` function.

A value of C for *locale* specifies the minimal C environment. A value of “ ” (space) specifies the implementation-defined native environment. Other implementation-defined strings can be passed as the second argument.

At program start-up, the equivalent of this call is executed:

```
setlocale(LC_ALL, "C");
```

Return Value

If *locale* is **NULL**, the locale is not changed and **setlocale()** returns the string associated with the specified category for the current locale.

If the selection can be honored, the specified category is changed and **setlocale()** returns the string associated with the new locale. If the selection cannot be honored, **setlocale()** returns **NULL**.

The strings returned are such that a subsequent call with the same string and category will restore that portion of the locale. The strings cannot be modified by the program and might be overwritten by subsequent calls to **setlocale()**.

System issues

The native locale is the C locale. No other locales are supported by this implementation.

See Also

[localeconv\(\)](#) — Query current locale for numeric format

[struct lconv](#) — Locale-specific numeric representation

Constants [LC_*](#) — Arguments to `setlocale()`

_set_matherr()

User defined math error function

Targets

`_set_matherr()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
int (*_set_matherr(
    int (*new_matherr)(struct exception *err_info) ))
    (struct exception *);
```

Description

`_set_matherr()` replaces the default `_matherr()` function with a user-defined error function `new_matherr` and returns the address of the previous `_matherr()` handler. This allows you to write a

function that temporarily replaces the current `_matherr()` handler, performs some math operations, and restores the previous `_matherr()` handler.

Function `new_matherr` is passed a pointer to a structure of type [exception](#), which contains information about the error detected. Function `new_matherr` might change the `retval` field of the [exception struct err_info](#).

Return Value

`_set_matherr()` returns the address of the previous `_matherr()` handler.

See Also

[matherr\(\), matherrx\(\) and _matherr\(\), _matherrx\(\)](#) — Provide standard error reporting for math functions

[struct exception](#) — Nature of a math error

_setmode()

Set stream mode (`_fmode`) to text or binary

Targets

`_setmode()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
void _setmode(FILE *stream, int mode);
```

Description

Some systems make a distinction between text streams and binary streams. On such systems, character sequences in text streams can be converted to other sequences on input or output while, for binary streams, I/O performs no conversions.

`_setmode()` has two distinct purposes:

- It can change the text/binary mode of `stream` to the mode specified by `mode`.
- It can set the variable `_fmode` to the mode specified by `mode`.

The possible values for `mode` are `_BINARY` (all streams default to binary) and `_TEXT` (all streams default to text).

Changing the mode of a stream

If `stream` is not `NULL`, it is assumed to point to a stream, which becomes a text stream if `mode` is `_TEXT`, or a binary stream if `mode` is `_BINARY`.

CAUTION If *stream* is not **NULL** but is not a valid **FILE** pointer, some arbitrary place in memory is overwritten.

Buffered output streams cannot be successfully changed between text and binary modes. To change the mode of an output stream *strm* in midstream, call [fflush\(\)](#) on *strm* prior to calling [_setmode\(\)](#) on *strm*.

Setting variable _fmode

Variable *_fmode* designates a stream as text or binary if no explicit designation is given when the stream is created — that is, when a file is opened. (To explicitly designate the text/binary mode of a stream when it is created, see [fopen\(\)](#).)

If *stream* is **NULL**, [_setmode\(\)](#) sets the variable *_fmode* to *mode*.

If [fopen\(\)](#) or [freopen\(\)](#) is called without specifying that the stream being returned is text or binary, the text/binary mode of the stream is set according to the value of *_fmode*.

Note Changing the value of *_fmode* has no effect on existing streams.

If you have licensed MetaWare C run-time library source code, you can specify the text/binary mode of the streams opened automatically at program start-up (`stdin`, `stdout`, and `stderr`). To do this, you modify the initial value of variable *_fmode* in library source file `_iob.c`, then link with the updated `_iob` object file.

You can also use the archiver to replace the `_iob` object file in the library so the changed value of *_fmode* persists for all future applications. For information about using the archiver, see the *ELF Linker and Utilities User's Guide*.

See Also

[File-Related Functions](#) on page 429

[typedef struct FILE — Information for controlling a stream](#)

Example

See the file `setmode.c` in the *install_dir/docs/examples/* directory on your distribution.

setvbuf()

Control file buffering

Targets

`setvbuf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
```

Description

setvbuf() controls the buffering of the stream pointed to by *stream*. It must be called after *stream* is associated with a file but before any I/O is performed.

The value of *mode* determines the type of buffering. [_IOFBF](#) and [_IOLBF](#) and [_IONBF](#) causes full buffering, [_IOFBF](#) and [_IOLBF](#) and [_IONBF](#) causes output to be line buffered, and [_IOFBF](#) and [_IOLBF](#) and [_IONBF](#) causes input/output to be unbuffered.

If *buf* is not **NULL**, the array it points to will be used instead of an automatically allocated one. The buffer size is determined by *size*. No assumptions can be made about the contents of the array at any time.

Return Value

If successful, **setvbuf()** returns 0 (zero). If not successful, it returns -1, indicating the request cannot be satisfied.

See Also

[File-Related Functions](#) on page 429

typedef struct FILE — Information for controlling a stream

Example

See the file *setvbuf.c* in the *install_dir/docs/examples/* directory on your distribution.

_setvect1()

Set low-priority interrupt to jump to target (ARC only)

Targets

_setvect1() works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
extern void _setvect1(int vector, _Interrupt1 void (*target)());
```

Description

For ARC targets only, **_setvect1()** sets low-priority interrupt vector *vector* to jump to handler *target*. The calling function must choose the appropriate *vector* for the priority level of the interrupt. For more information, see the *Programmer's Reference Manual* for your ARC processor.

For information about the macro **_Interrupt1** for declaring level-one interrupt handlers, see the *MetaWare C/C++ Programmer's Guide for ARC*.

See Also

[_setvect2\(\)](#) — Set mid- or high-priority interrupt to jump to target (ARC only)

_setvect2()

Set mid- or high-priority interrupt to jump to target (ARC only)

Targets

_setvect2() works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
extern void _setvect2(int vector, _Interrupt2 void (*target)());
```

Description

For ARC targets only, **_setvect2()** sets mid- or high-priority interrupt vector *vector* to jump to handler *target*. The calling function must choose the appropriate *vector* for the priority level of the interrupt. For more information, see the *Programmer's Reference Manual* for your ARC processor.

For information about the macro **_Interrupt2** for declaring level-two interrupt handlers, see the *MetaWare C/C++ Programmer's Guide for ARC*.

See Also

[_setvect1\(\)](#) — Set low-priority interrupt to jump to target (ARC only)

signal()

Set up a signal handler

Targets

signal() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <signal.h>    /* Required */
void (*signal(int sig, void (*func)(int)))(int);
```

Description

`signal()` arranges for *func* to be used as the signal handler for the signal *sig*.

- If *func* is [SIG_DFL](#) and [SIG_ERR](#) and [SIG_IGN](#), the signal is ignored.
- If *func* is [SIG_DFL](#), the signal is handled in the default manner.
- Otherwise, *func* should point to a function to be called when *sig* is received.

When a signal *sig* is received, if the signal handler for *sig* is not [SIG_IGN](#) or [SIG_DFL](#), *sig* is handled as follows:

1. The signal handler for *sig* is reset to the default — which is equivalent to executing `signal(sig, SIG_DFL)`.
2. The signal handler for *sig* is called with *sig* as its argument: the equivalent of `(*func)(sig)` is executed.
3. If *func* returns — that is, if *func* does not call `abort()`, `exit()`, or `longjmp()` (see caution) — execution resumes at the point at which the signal was received (unless *sig* was [SIGFPE](#); then the behavior is undefined).

The equivalent of `signal(sig, SIG_DFL)` is executed for each signal at program start-up.

CAUTION If the compiler generates a call to any of the functions `_kill()`, `raise()`, or `signal()`, the library module `signal.obj` is linked into the executable.

This module provides a set of interrupt handlers markedly different from those supplied by the operating system.

If no calls to any of these functions are generated in a program, the routines used to handle interrupts are determined by the operating system.

Keep this in mind when you use interrupt routines in a program that calls one of these functions.

Because signal-handler routines are called asynchronously when an interrupt occurs, the process will likely be left in an undefined state. If the signal was not explicitly called by `raise()`, certain precautions need to be taken.

Do not do any of the following:

- issue I/O requests to the console or a disk (`conio`, `read()`)
- call memory allocation routines (`malloc()`, `putenv()`)
- use any functions that call BIOS or DOS (`clock()`, `stat()`)
- use `longjmp()` to return control to the process

In addition, limited stack space is available to the signal handler routine. Recursion or similar stack usage might overwrite data. This warning does not apply when the signal handler is called by `raise()`.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

NOTE The following system issues are specific to DOS and Windows NT targets.

Child processes invoked by **exec***() or **spawn***() have all signals reset to the default; signal handlers are local to the current process.

On DOS, many of the signals listed in `signal.h` have no meaning. Only **SIGABRT**, **SIGFPE**, and **SIGINT** normally occur. Use **raise()** to invoke other signals.

Floating-Point Exceptions

SIGFPE is generated by floating-point exceptions and by the integer divide-by-zero exception. When a **SIGFPE** occurs, the signal handler is invoked with an extra parameter called **sub_code**. For example, the following is a prototype of a signal handler that expects to catch **SIGFPE**:

```
void sighandler(int signal_number, int sub_code);
```

Floating-point exceptions are normally disabled. To enable exceptions, use function **_control87()**. Floating-point exceptions are supported only if you did not compile your program with option **-fsoft**.

The floating-point subcodes declared in `signal.h` have the following values and meanings:

- _FPE_DENORMAL** — Denormalized operand exception
- _FPE_ZERODIVIDE** — Divide-by-zero exception
- _FPE_OVERFLOW** — Overflow exception
- _FPE_UNDERFLOW** — Underflow exception
- _FPE_INEXACT** — Precision exception
- _FPE_STACKOVERFLOW** — Stack overflow exception
- _FPE_STACKUNDERFLOW** — Stack underflow exception
- _FPE_EXPLICITGEN** — Signal handler invoked via **raise()**
- _FPE_INTDIV0** — Integer divide-by-zero exception

For **SIGFPE**, if you use **SIG_IGN** or **SIG_ERR** in place of *func*, **signal()** returns **SIG_ERR**. If you use **SIG_ERR** for any other signal, **raise()** returns -1 (**SIG_ERR**) when attempting to send that signal. If **SIG_IGN** catches a hardware signal, the resulting behavior is undefined.

Return Value

If successful, **signal()** returns a pointer to the previous handler for the specified signal. If not successful, **signal()** returns **SIG_ERR** and sets global variable `errno` to **EINVAL** ([Invalid argument given](#)).

See Also

[**raise\(\)** — Raise a signal](#)

[**SIG_DFL**, **SIG_DFL and SIG_ERR** and **SIG_IGN**, **SIG_DFL and SIG_ERR and SIG_IGN** — Macros; arguments to **signal\(\)**](#)

[**SIG*** — Arguments to **signal\(\)** and **raise\(\)**](#)

Example

See the file `signal.c` in the *install_dir/docs/examples/* directory on your distribution.

sin()

Sine

Targets

sinf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double sin(double x);
```

Description

sinf() computes the sine of x , where x is in radians. A large magnitude number might yield a result with little or no significance. **sinf()** might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

sinf() returns the sine of x , where x is in radians.

See Also

[asin\(\)](#) — Arc sine

[asinh\(\)](#) and [asinhf\(\)](#) — Hyperbolic arc sine

[cos\(\)](#) — Cosine

[sinf\(\)](#) — Sine using single-precision floating point

[sinh\(\)](#) — Hyperbolic sine

[sinhf\(\)](#) — Hyperbolic sine using single-precision floating point

[tan\(\)](#) — Tangent

[tanf\(\)](#) — Tangent using single-precision floating point

[Transcendental Math Functions](#) on page 424

Example

See the file `sinf.c` in the `install_dir/docs/examples/` directory on your distribution.

sinf()

Sine using single-precision floating point

Targets

sinf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
float sinf(float x);
```

Description

sinf() computes the sine of x , where x is in radians. A large magnitude number might yield a result with little or no significance. **sinf()** might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

sinf() returns the sine of x , where x is in radians.

See Also

[asin\(\)](#) — Arc sine

[asinf\(\)](#) — Arc sine using single-precision floating point

[asinh\(\) and _asinh\(\)](#) — Hyperbolic arc sine

[cos\(\)](#) — Cosine

[cosf\(\)](#) — Cosine using single-precision floating point

[sinh\(\)](#) — Hyperbolic sine

[sinhf\(\)](#) — Hyperbolic sine using single-precision floating point

[tan\(\)](#) — Tangent

[tanf\(\)](#) — Tangent using single-precision floating point

[Transcendental Math Functions](#) on page 424

sinh()

Hyperbolic sine

Targets

sinh() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double sinh(double x);
```

Description

sinh() computes the hyperbolic sine of x . Theoretically, **sinh()** has an unbounded domain and range. However, if the magnitude of x is too large, such that **sinh(x)** cannot be represented, **sinh()** sets global

variable `errno` to **ERANGE** and returns **HUGE_VAL** with the same sign as x . **sinh()** might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

sinh() returns the hyperbolic sine of x .

See Also

[asin\(\)](#) — Arc sine

[asinh\(\) and _asinh\(\)](#) — Hyperbolic arc sine

[cos\(\)](#) — Cosine

[cosf\(\)](#) — Cosine using single-precision floating point

[sin\(\)](#) — Sine

[sinf\(\)](#) — Sine using single-precision floating point

[sinhf\(\)](#) — Hyperbolic sine using single-precision floating point

[tan\(\)](#) — Tangent

[tanf\(\)](#) — Tangent using single-precision floating point

[tanh\(\)](#) — Hyperbolic tangent

[tanhf\(\)](#) — Hyperbolic tangent using single-precision floating point

[Transcendental Math Functions](#) on page 424

Global variable [errno](#) — An int used to record error numbers

Example

See the file `sinh.c` in the `install_dir/docs/examples/` directory on your distribution.

sinhf()

Hyperbolic sine using single-precision floating point

Targets

sinhf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
float sinhf(float x);
```

Description

sinhf() computes the hyperbolic sine of x . Theoretically, **sinhf()** has an unbounded domain and range. However, if the magnitude of x is too large, such that **sinhf(x)** cannot be represented, **sinhf()** sets

global variable `errno` to **ERANGE** and returns **HUGE_VAL** with the same sign as *x*. `sinhf()` might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

`sinhf()` returns the hyperbolic sine of *x*.

See Also

[asin\(\)](#) — Arc sine

[asinf\(\)](#) — Arc sine using single-precision floating point

[asinh\(\)](#) and [_asinh\(\)](#) — Hyperbolic arc sine

[cos\(\)](#) — Cosine

[cosf\(\)](#) — Cosine using single-precision floating point

[sin\(\)](#) — Sine

[sinf\(\)](#) — Sine using single-precision floating point

[sinh\(\)](#) — Hyperbolic sine

[tan\(\)](#) — Tangent

[Transcendental Math Functions](#) on page 424

Global variable `errno` — An int used to record error numbers

_sleep()

Temporarily suspend program execution

Targets

`_sleep()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>
void _sleep(unsigned seconds)
```

Description

`_sleep()` suspends the execution of the current program for the amount of time specified by *seconds*. The interval is accurate to the nearest hundredth of a second or to the system clock interval, whichever is greater.

Example

See the file `sleep.c` in the *install_dir/docs/examples/* directory on your distribution.

spawn*() and _spawn*()

Execute a child process

Targets

The `spawn*()` and `_spawn*()` functions work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The `spawn*()` functions are Non-ANSI; the `_spawn*()` functions are Extra-ANSI.

Reentrancy

The `spawn*()` and `_spawn*()` functions are Not-Reentrant.

Synopsis

```
#include <process.h>

int spawnl (mode, path, arg0,..., argN, NULL);
int spawnle (mode, path, arg0,..., argN, NULL, envp);
int spawnlp (mode, path, arg0,..., argN, NULL);
int spawnlpe(mode, path, arg0,..., argN, NULL, envp);

int spawnv (mode, path, argv);
int spawnve (mode, path, argv, envp);
int spawnvp (mode, path, argv);
int spawnvpe(mode, path, argv, envp);

int _spawnl (mode, path, arg0,..., argN, NULL);
int _spawnle (mode, path, arg0,..., argN, NULL, envp);
int _spawnlp (mode, path, arg0,..., argN, NULL);
int _spawnlpe(mode, path, arg0,..., argN, NULL, envp);

int _spawnv (mode, path, argv);
int _spawnve (mode, path, argv, envp);
int _spawnvp (mode, path, argv);
int _spawnvpe(mode, path, argv, envp);
```

Description

The `spawn*()` functions execute child processes. The functions vary in the following ways:

- how the parent process locates the child
- the format in which the parent passes command-line arguments to the child
- whether the child inherits the parent's environment or the parent passes the child a new environment

These differences are specified by the letters at the end of the function name.

Suffix **p**: The function uses the PATH environment variable to locate the file to be executed.

Suffix **I**: Each command-line argument is passed separately.

Suffix **v**: The command-line arguments are passed as an array of pointers.

Suffix **e**: An environment is passed as an array of pointers to environment strings.

The mode Argument

The *mode* argument determines how the program is loaded and how the calling program behaves. *mode* can be one of the P * values defined in process.h (P_DETACH, P_NOWAIT, P_NOWAITO, P_OVERLAY, or P_WAIT).

NOTE On DOS, only P_WAIT is currently implemented.

The path Argument, Suffix “p”

path specifies the file to be executed. It can be the name of the file, the path from the current working directory, or the full pathname.

If the file name does not have an extension and does not end with a period, the **spawn*()** or **_spawn*()** function first attempts to locate the file name as is, in the current working directory. If the search is unsuccessful, the function appends .exe to the file name and tries again.

If the specified file is not found in the current working directory, those **spawn*()** and **_spawn*()** functions whose name includes the p suffix use the PATH environment variable to locate the child process.

The argN and argv[N] Arguments, Suffixes “l” and “v”

The command-line argument(s) passed by the parent to the child process might be listed separately or might consist of an array of pointers. At least one argument must be passed to the child process. This argument, *arg0* (or *argv[0]*), is usually a copy of the *path* argument.

Each command-line argument must be a NUL-terminated character string not longer than 128 bytes.

NOTE On DOS, the combined total length of all the command-line arguments cannot exceed 128 bytes.

Those **spawn*()** and **_spawn*()** functions whose name includes the l suffix all pass their command-line arguments separately (as *arg0*, *arg1*, ..., *argN*).

Those **spawn*()** and **_spawn*()** functions whose name includes the v suffix pass an array of pointers to command-line arguments (as *argv[0]*, *argv[1]*, ..., *argv[N]*).

The envp Argument, Suffix “e”

Child processes executed by those **spawn*()** and **_spawn*()** functions whose name does not include the e suffix inherit the environment of the parent process.

Child processes executed by those **spawn*()** and **_spawn*()** functions whose name includes the e suffix allow the parent to pass any arbitrary environment to the child through the *envp* argument.

envp must be an array of pointers to NUL-terminated strings. These strings usually define environment variables such as the following:

```
PATH=c:\bin;c:\highc
LIB=c:\highc\lib
```

The last pointer in this array must be the NULL pointer.

If *envp* is passed as **NULL**, then those **spawn*()** and **_spawn*()** functions whose name includes the e suffix pass a copy of the parent's environment to the child (as do the other **spawn*()** and **_spawn*()** functions).

Return Value

If *mode* is **_P_WAIT**, the **spawn***() and **_spawn***() functions return the exit status of the child process. If the child process terminated normally, the exit status is 0 (zero). If the child process terminated abnormally, the exit status is greater than zero. If the child process could not be started, the functions return -1 and set global variable **errno** to one of the following values:

E2BIG — [Argument list too long](#)

EINVAL — [Invalid argument given](#)

ENOENT — [File or directory not found; pathname element does not exist](#)

ENOEXEC — [EXEC format error](#)

ENOMEM — [Not enough memory, or invalid DOS arena headers](#)

NOTE Setting **errno** to **E2BIG** can indicate that the argument list exceeds 128 bytes, or the environment information exceeds 32K (DOS only).

If *mode* is not **_P_WAIT**, each function returns the process ID of the child process.

System issues

On DOS, executing a child process from within a parent process requires enough conventional memory (memory below 640K) to be available for running the child executable. If sufficient conventional memory is not available, the results are undefined. Conventional memory can be controlled with Phar Lap 386|DOS-Extender options **-MAXREAL** and **-MINREAL**. See the *Phar Lap 386|DOS-Extender Reference Manual*.

See Also

[abort\(\)](#) — Terminate a program abnormally

[atexit\(\)](#) — Register a function for execution at program termination

[exit\(\)](#) — Terminate a program normally

[system\(\)](#) — Pass a command to the operating system

Example

See the file *spawn.c* in the *install_dir/docs/examples/* directory on your distribution.

_splitpath()

Split a pathname into its components

Targets

_splitpath() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>          /* Required */
void _splitpath(const char *path, char *drive,
    char *dirpath, char *filename, char *extension);
```

Description

`_splitpath()` decomposes a full pathname *path* into *drive*, *dirpath*, *filename*, and *extension*, the four components of any full pathname.

The result of `_splitpath()` is defined as follows:

- If *path* has a drive specification, *drive* contains the drive letter followed by a colon (:). Otherwise, *drive* contains the empty string.
- If *path* has a directory path, *dirpath* contains the directory path including the trailing backslash, if any. Otherwise, *dirpath* contains the empty string.
- *filename* contains the base file name without any file extension.
- If *path* has a file extension, *extension* contains the file extension including the leading period. Otherwise, *extension* contains the empty string.

The maximum size necessary for each buffer is specified by the [MAX_*](#) constants declared in `stdlib.h`.

For more information about the [MAX_*](#) constants, see [Constants and Globals](#) on page 319.

See Also

[makepath\(\)](#) — [Create a path string from components](#)

Constants [MAX_*](#) — [Maximum length of several pathname components](#)

Example

See the file `splitpat.c` in the *install_dir/docs/examples/* directory on your distribution.

sprintf()

Print to a string

Targets

`sprintf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdio.h>
int sprintf(char *str, const char *format, ...);
```

Description

`sprintf()` is equivalent to `printf(format, ...)`, except that the output is written to the string *str* rather than to `stdout`; see `printf()` for a detailed explanation.

CAUTION Garbage might be printed if the number of arguments passed to **sprintf()** is less than the number specified by *format*, or if the types of the arguments to **sprintf()** do not match the types specified by *format*.

Return Value

sprintf() returns the number of characters printed, excluding the terminating NUL character ('\0').

See Also

[printf\(\) — Print to stdout](#)

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

Example

See the file *sprintf.c* in the *install_dir/docs/examples/* directory on your distribution.

sqrt()

Square root

Targets

sqrt() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

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

Description

sqrt() calculates the non-negative square root of *x*. If *x* < 0.0, **sqrt()** returns 0.0 and sets global variable *errno* to [EDOM \(Mathematical domain error\)](#). **sqrt()** might be inlined when the toggle Recognize_library is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

sqrt() returns the non-negative square root of *x*.

See Also

[sqrtf\(\) — Square root using single-precision floating point](#)

sqrtf()

Square root using single-precision floating point

Targets

sqrtf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
float sqrtf(float x);
```

Description

sqrts() calculates the non-negative square root of *x*. If *x* < 0.0, **sqrts()** returns 0.0 and sets global variable *errno* to [EDOM](#) ([Mathematical domain error](#)). **sqrts()** might be inlined when the toggle Recognize_Library is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

sqrts() returns the non-negative square root of *x*.

See Also

[sqrt\(\)](#) — [Square root](#)

srand()

Seed the pseudo-random number generator

Targets

srand() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>
void srand(unsigned int seed);
```

Description

srand() uses its argument as a seed for the sequence of pseudo-random numbers generated by **rand()**. If **rand()** is called before any calls to **srand()**, the seed used is 1 (one).

See Also

[rand\(\)](#) — [Generate a pseudo-random number](#)

_srotl() and _srotr()

Rotate a **short int** left or right

Targets

The **_srot***() functions work on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

Both functions are Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h>
unsigned short _srotl(unsigned short val,int bits);
unsigned short _srotr(unsigned short val,int bits);
```

Description

_srotl() rotates *val* to the left by the number of bits given in *bits*.

_srotr() rotates *val* to the right by *bits* bits.

Return Value

_srotl() returns *val* rotated to the left.

_srotr() returns *val* rotated to the right.

See Also

[**_crotl\(\)** and **_crotr\(\)**, **_crotl\(\)** and **_crotr\(\)**](#) — Rotate a char left or right

[**_lrotl\(\)** and **_lrotr\(\)**, **_lrotl\(\)** and **_lrotr\(\)**](#) — Rotate a long int left or right

[**_rotl\(\)** and **_rotr\(\)**, **_rotl\(\)** and **_rotr\(\)**](#) — Rotate an int left or right

Example

See the file *srot.c* in the *install_dir/docs/examples/* directory on your distribution.

sscanf()

Read values from a string

Targets

sscanf() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdio.h>
int sscanf(const char *str, const char *format, ...);
```

Description

sscanf() is identical to **scanf(format, ...)**, except that the input is read from the string *str* rather than from **stdin**. See **scanf()** for a detailed explanation.

CAUTION If fewer arguments are passed to **sscanf()** than specified by *format*, arbitrary places in memory are overwritten with the results of the specified conversions.

sscanf() interprets each argument that follows *format* as a pointer to the type specified by the corresponding conversion specification. If an argument is not the expected pointer, it is nonetheless treated as if it were, potentially causing some arbitrary place in memory to be overwritten with the result of the conversion. If the incorrect argument is not the same size as a pointer, subsequent arguments might be misinterpreted as well.

Return Value

sscanf() returns the number of input values assigned, or **EOF** if end-of-file is encountered before the first conflict or conversion. If no conflict occurs, **sscanf()** returns when the end of the format string is encountered.

See Also

[scanf\(\)](#) — Read values from **stdin**

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

Example

See the file *sscanf.c* in the *install_dir/docs/examples/* directory on your distribution.

stat() and _stat()

Get information about a file or directory

Targets

stat() and **_stat()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

stat() is Non-ANSI; **_stat()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <sys\types.h>    /* Required */
#include <sys\stat.h>     /* Required */
int stat(char *path, struct stat *buffer);
int _stat(char *path, struct stat *buffer);
```

Description

stat() and **_stat()** gather information about the file or directory specified by *path* and store it in the **stat** structure pointed to by *buffer*. The **stat** structure is defined in `sys\stat.h`.

Return Value

If the file information is successfully obtained, **stat()** and **_stat()** return 0 (zero). If not successful, the functions return -1 and set global variable `errno` to **ENOENT** ([File or directory not found; pathname element does not exist](#)).

See Also

[File-Related Functions](#) on page 429

[struct stat — Information about a file](#)

Example

See the file `stat.c` in the *install_dir/docs/examples/* directory on your distribution.

strcat()

Concatenate two strings

Targets

strcat() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */
char *strcat(char *s1, const char *s2);
```

Description

strcat() appends a copy of the string *s2* to the string *s1*. The first character of *s2* writes over the NUL character ('\0') that ends *s1*.

CAUTION **strcat()** writes until the end of string *s2* is encountered, without regard for the size of *s1*.

Return Value

strcat() returns *s1*.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

See Also

[String-Handling Functions](#) on page 432 ([Concatenating or Appending Strings](#))

Example

See the file `strcat.c` in the *install_dir/docs/examples/* directory on your distribution.

_strcats()

Concatenate multiple strings up to a limited number of characters

Targets

_strcats() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */
char *_strcats(int len, char *s1, const char *s2, ..., NULL);
```

Description

_strcats() concatenates two or more strings. If *s2* is **NULL**, nothing happens. If *s2* is not **NULL**, strings *s2* and subsequent arguments are concatenated onto *s1* such that *s1* has at most *len*-1 non-NUL characters (where NUL is '\0').

The final argument to **_strcats()** must be **NULL**; this terminates the concatenation process. A final NUL character is appended to *s1* if any characters at all were appended.

The first character of *s2* writes over the NUL character that ends *s1*, and similarly for each subsequent argument. If *s1* initially does not contain a NUL character, the results are undefined.

CAUTION **_strcats()** has no regard for the actual size of *s1*. You must ensure that *s1* is large enough to hold the strings to be concatenated. The *len* argument is provided for this purpose.

Return Value

_strcats() returns a pointer to *s1*.

See Also

[String-Handling Functions](#) on page 432 ([Concatenating or Appending Strings](#))

Example

See the file `strcats.c` in the *install_dir/docs/examples/* directory on your distribution.

strchr()

Find the first occurrence of a character in a string

Targets

strchr() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */
char *strchr(const char *s, int c);
```

Description

strchr() finds the first occurrence of the character *c* (the least significant byte of the **int** *c*) in the string *s*. NUL is a valid value for *c* and results in **strchr()** returning a pointer to the end of the string (to the terminating NUL).

strchr() takes *c* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to **strchr()**, it is automatically coerced to type **int**.

Return Value

strchr() returns a pointer to the character if it is found. If not successful, it returns **NULL**.

System issues

On UNIX, this function can be provided by the system library.

See Also

[**memchr\(\)** — Find a character in an area of memory](#)

[String-Handling Functions](#) on page 432 ([Determining String Length](#) and [Finding Characters in Strings](#))

Example

See the file *strchr.c* in the *install_dir/docs/examples/* directory on your distribution.

strcmp()

Compare one string to another

Targets

strcmp() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

Description

strcmp() compares the string *s1* to the string *s2*.

Note If two strings of unequal length compare equal to the extent of the shorter string, the longer string is lexicographically greater; for instance, "abcd" compares greater than "ab".

Return Value

This is the result of the comparison:

- If *s1* is less than *s2*, **strcmp()** returns an **int** less than 0 (zero).
- If *s1* is greater than *s2*, **strcmp()** returns an **int** greater than 0 (zero).
- If *s1* is equal to *s2*, **strcmp()** returns 0 (zero).

System issues

On UNIX, this function can be provided by the system library.

See Also

[memcmp\(\)](#) — Compare two areas of memory

[String-Handling Functions](#) on page 432 ([Comparing Strings](#))

Example

See the file `strcmp.c` in the *install_dir/docs/examples/* directory on your distribution.

strcmpi() and _strcmpi()

Compare strings, ignoring case

Targets

strcmpi() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_strcmpi() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

strcmpi() is Non-ANSI; **_strcmpi()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>          /* Required */
int strcmpi(const char *s1, const char *s2);
int _strcmpi(const char *s1, const char *s2);
```

Description

strcmpi() and **_strcmpi()** compare the characters of the two strings, *s1* and *s2*. The compare considers uppercase and lowercase alphabetic letters to be equal.

strcmpi() is identical to [stricmp\(\) and _stricmp\(\)](#), and **_strcmpi()** is identical to [stricmp\(\) and _stricmp\(\)](#).

NOTE If two strings of unequal length compare equal to the extent of the shorter string, the longer string is lexicographically greater; for instance, "abcd" compares greater than "ab".

Return Value

This is the result of the comparison:

- If $s1$ is less than $s2$, **strcmpl()** and **_strcmpl()** return an **int** less than 0 (zero).
- If $s1$ is greater than $s2$, **strcmpl()** and **_strcmpl()** return an **int** greater than 0 (zero).
- If $s1$ is equal to $s2$, **strcmpl()** and **_strcmpl()** return 0 (zero).

System issues

On UNIX, this function can be provided by the system library.

See Also

[String-Handling Functions](#) on page 432 ([Comparing Strings](#))

Example

See the file `strcmpl.c` in the *install_dir/docs/examples/* directory on your distribution.

strcoll()

Compare strings based on current locale

Targets

strcoll() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>
int strcoll(const char *s1, const char *s2);
```

Description

strcoll() compares the string $s1$ to the string $s2$, using the collating sequence of the program's locale. MetaWare C supports only the standard locale, so **strcoll()** is equivalent to [strcmpl\(\)](#).

Return Value

This is the result of the comparison:

- If $s1$ is less than $s2$, **strcoll()** returns an **int** less than 0 (zero).
- If $s1$ is greater than $s2$, **strcoll()** returns an **int** greater than 0 (zero).
- If $s1$ is equal to $s2$, **strcoll()** returns 0 (zero).

See Also

[String-Handling Functions](#) on page 432 ([Comparing Strings](#) and [Converting, Transforming, or Formatting Strings](#))

strcpy()

Copy the characters of one string into another

Targets

strcpy() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */
char *strcpy(char *dest, const char *source);
```

Description

strcpy() copies the characters of string *source* into string *dest*, including the terminating NUL character ('\0'). strcpy() copies from left to right (low addresses to high addresses).

If *source* < *dest* < *source* + strlen(*source*), that is, if the source and destination areas overlap, with the destination area beginning within the source area, use [_rstrcpy\(\)](#).

strcpy() might be inlined when the toggle Recognize_Library is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

CAUTION **strcpy()** writes until the end of string *source* is encountered, without regard for the length of *dest*. If the source and destination areas overlap such that *source* < *dest* < *source* + strlen(*source*), the behavior is undefined.

Return Value

strcpy() returns a pointer to *dest*.

Surprises

strcpy() copies the second argument to the first.

System issues

On UNIX, this function can be provided by the system library.

See Also

[memcpy\(\)](#) — [Copy from one place in memory to another](#)

[memset\(\)](#) — [Duplicate a character across an area of memory](#)

[_rmemcpy\(\)](#) — [Copy from one place in memory to another](#)

[String-Handling Functions](#) on page 432 ([Copying Strings](#))

Example

See the file `strcpy.c` in the *install_dir/docs/examples/* directory on your distribution.

strcspn()

Determine the length of the prefix of a string not containing any characters from another string

Targets

`strcspn()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */
size_t strcspn(const char *s1, const char *s2);
```

Description

`strcspn()` finds the length of that segment of the string *s1* that starts at the beginning of the string and is made up entirely of characters that do not occur in the string *s2*. NUL is not considered a part of *s2* for this purpose.

Return Value

`strcspn()` returns the length described previously.

See Also

[memchr\(\)](#) — Find a character in an area of memory

[String-Handling Functions](#) on page 432 ([Finding Characters in Strings](#) and [Determining String Length](#))

Example

See the file `memset.c` in the *install_dir/docs/examples/* directory on your distribution.

_strdate()

Convert the current date to a string

Targets

`_strdate()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <time.h>          /* Required */
char *_strdate(char *sptr);
```

Description

`_strdate()` obtains the current date, converts it to character format, and places it in the string pointed to by *sprt*.

The date is formatted as *mm/dd/yy*, where *mm* is the month, *dd* is the day, and *yy* is the year. Each part of the date is two digits, and the string that *sprt* points to must be at least nine bytes long. The function places a terminating NUL character ('\0') at the end of the date string.

Return Value

`_strdate()` returns a pointer to *sprt*.

See Also

[String-Handling Functions](#) on page 432 ([Converting Other Data to Strings](#))

[Time-Related Functions](#) on page 434

Example

See the file `strdate.c` in the *install_dir/docs/examples/* directory on your distribution.

strup() and _strup()

Make a copy of a string in the heap

Targets

`strup()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_strup()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`strup()` is Non-ANSI; `_strup()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <string.h>           /* Required */
char *strup(const char *strptr);
char *_strup(const char *strptr);
```

Description

`strup()` and `_strup()` compute the length of the string pointed to by *strptr*, call `malloc()` to obtain adequate space from the heap, then copy *strptr* into the space obtained from `malloc()`.

Return Value

If successful, `strup()` and `_strup()` return a pointer to the new copy of the string. If `malloc()` could not obtain enough space for the new copy of the string, `strup()` and `_strup()` return `NULL`.

See Also

[malloc\(\)](#) — [Dynamically allocate uninitialized storage](#)

[String-Handling Functions](#) on page 432 ([Copying Strings](#))

Example

See the file `strdup.c` in the `install_dir/docs/examples/` directory on your distribution.

strerror() and `_strerror()`

Map error code; append error-message string to user message

Targets

`strerror()` and `_strerror()` work on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

`strerror()` is ANSI compliant; `_strerror()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>    /* Required          */
#include <stdio.h>      /* Required for variables  */
int errno;                /* Error number        */
int sys_nerr;             /* Number of error message */
char *sys_errlist[sys_nerr];

char *strerror(int errnum);
char *_strerror(char *string);
```

Description

`strerror()` uses `errnum` to index into `sys_errlist[]`, the global array of error-message strings.

`strerror()` copies the error-message string into a static buffer and returns a pointer to the buffer. This buffer is overwritten by subsequent calls to `strerror()`.

NOTE All error-message strings in `sys_errlist[]` are terminated by newline ('\n') characters.

`_strerror()` returns a pointer to a string that consists of the following:

`string: Error_message_string`

`Error_message_string` is the error-message string in `sys_errlist[]` that corresponds to the value in the global variable `errno`. This string is created in a static buffer of length 128. To avoid exceeding the bounds of the buffer, the message in `string` must be less than 95 characters long.

If `string` is passed as `NULL`, the pointed-to string does not include the “:`”`.

Global variable `errno` contains the error code for the last library call that produced an error.

Return Value

`strerror()` and `_strerror()` return a pointer to an error-message string.

System issues

The error-message strings returned by **strerror()** and **_strerror()** do not contain newlines. If your application runs on DOS or Windows NT and you want the error message string to contain newlines, you can do either of the following:

- Set global variable `_strerror_nl` to 1 (one) as part of your program initialization.
- Link in an object file that defines `_strerror_nl` and preinitializes it to 1 (one), as in:
`char _strerror_nl = 1;`

See Also

[clearerr\(\)](#) — Clear end-of-file and error flags

[ferror\(\)](#) — Test for a read or write error on a file

[perror\(\)](#) — Print an error message

Example

See the file `strerror.c` in the `install_dir/docs/examples/` directory on your distribution.

strftime()

Format time and date string

Targets

strftime() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <time.h> /* Required */
size_t strftime(char *s, size_t max, const *format,
               const struct tm *timeptr);
```

Description

strftime() formats a string *s* according to the specifications in string *format*. The format string can consist of zero or more conversion specifications and ordinary multibyte characters.

A conversion specification consists of a ‘%’ character followed by a character that determines the conversion. Ordinary multibyte characters in the format string are copied unchanged into *s*.

No more than *max* characters (including the terminating null character) are placed in *s*. If *format* and *s* overlap, the behavior is undefined.

The conversion specifications are replaced as follows:

%a — Locale’s abbreviated weekday name

%A — Locale’s full weekday name

%b — Locale’s abbreviated month name

%B — Locale's full month name
%c — Locale's appropriate date and time representation
%d — Day of the month as a decimal number (01 — 31)
%H — Hour (24-hour clock) as a decimal number (00 — 23)
%I — Hour (12-hour clock) as a decimal number (01 — 12)
%j — Day of the year as a decimal number (001 — 366)
%m — Month as a decimal number (01 — 12)
%M — Minute as a decimal number (00 — 59)
%p — Locale's equivalent of either AM or PM
%S — Second as a decimal number (00 — 59)
%U — Week of the year (Sunday as first day of the week) as a decimal number (00 — 51)
%w — Day of the week (Sunday first) as a decimal number (0 — 6)
%W — Week of the year (Monday as first day of the week) as a decimal number (00 — 51)
%x — Locale's appropriate date representation
%X — Locale's appropriate time representation
%y — Year without century as a decimal number (00 — 99)
%Y — Year with century as a decimal number
%Z — Time-zone name (or nothing if time zone is undeterminable)
%% — Replaced by ‘%’

If a conversion specification is not one of these, the behavior is undefined.

Return Value

If successful, **strftime()** returns the length of *s*. If the length of *s* exceeds *max*, **strftime()** returns 0 (zero).

See Also

[String-Handling Functions](#) on page 432 ([Converting, Transforming, or Formatting Strings](#))

[Time-Related Functions](#) on page 434

strcmp() and _strcmp()

Compare strings, ignoring case

Targets

strcmp() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_stricmp()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`strcmp()` is Non-ANSI; `_stricmp()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>          /* Required */
int strcmp(const char *s1, const char *s2);
int _stricmp(const char *s1, const char *s2);
```

Description

See [_stcmpi\(\)](#) and [_strcmpi\(\)](#) and [_strempi\(\)](#) and [_strcmpni\(\)](#).

strlen()

Determine the length of a string

Targets

`strlen()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */
size_t strlen(const char *s);
```

Description

`strlen()` determines the number of characters in the string *s*, not counting the terminating NUL character ('\0'). `strlen()` might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

`strlen()` returns the length of the string *s*.

System issues

On UNIX, this function can be provided by the system library.

See Also

[String-Handling Functions](#) on page 432 ([Determining String Length](#))

Example

See the file `strlen.c` in the `install_dir/docs/examples/` directory on your distribution.

strlwr() and _strlwr()

Make string all lowercase

Targets

`strlwr()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_strlwr()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`strlwr()` is Non-ANSI; `_strlwr()` is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>           /* Required */
char *strlwr(char *strptr);
char *_strlwr(char *strptr);
```

Description

`strlwr()` and `_strlwr()` convert all uppercase alphabetic letters in the string pointed to by `strptr` into their equivalent lowercase letters. Other characters are not modified.

Return Value

`strlwr()` and `_strlwr()` return `strptr`.

System issues

On UNIX, this function can be provided by the system library.

See Also

[String-Handling Functions](#) on page 432 ([Converting, Transforming, or Formatting Strings](#))

Example

See the file `strlwr.c` in the `install_dir/docs/examples/` directory on your distribution.

strncat() and _strncat()

Append characters of one string to another, up to some limit n

Targets

`strncat()` and `_strncat()` work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`strncat()` is ANSI compliant; `_strncat()` is Extra-ANSI.

Reentrancy

Reentrant

Synopsis

```
#include <string.h>           /* Required */
char *strncat(char *s1, const char *s2, size_t n);
char *_strncat(char *s1, const char *s2, size_t n);
```

Description

strncat() appends up to *n* characters from the string *s2* to the string *s1*.

Similarly, **_strncat()** appends characters from the string *s2* to the string *s1*, up to a total size limit for *s1* of *n* characters.

With both functions, the first character of *s2* writes over the NUL character ('\0') that ends *s1*, and the result is always NUL-terminated. This means that, for **strncat()**, *s1* must have at least *n* bytes following its terminating NUL. For **_strncat()**, the resulting string *s1* has a total of up to *n*-1 non-NUL characters.

Note **_strncat()** differs from **strncat()** in that the third argument to **_strncat()** is the maximum length for the result *s1*, while the third argument to **strncat()** is the maximum number of characters to be appended to *s1*. **strncat()** appends up to *n* characters without regard for the size of *s1*.

Return Value

strncat() and **_strncat()** return a pointer to the concatenated string *s1*.

System issues

On UNIX, this function can be provided by the system library.

See Also

[String-Handling Functions](#) on page 432 ([Concatenating or Appending Strings](#))

Example

For **strncat()**, see the file **strncat.c** in the *install_dir/docs/examples/* directory on your distribution.

For **_strncat()**, see the file **strncat2.c** in the *install_dir/docs/examples/* directory on your distribution.

strcmp()

Compare characters of one string to characters of another

Targets

strcmp() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */
int strcmp(const char *s1, const char *s2, size_t n);
```

Description

strcmp() compares the string *s1* to the string *s2*. No more than *n* characters are compared.

NOTE If two strings of unequal length compare equal to the extent of the shorter string and the length of the shorter string is less than *n*, the longer is lexicographically greater; for instance, "abc" compares greater than "ab".

Return Value

This is the result of the comparison:

- If the first *n* characters of *s1* (or the entire string, if it contains less than *n* characters) compare greater than the corresponding characters from *s2*, **strcmp()** returns an **int** greater than 0 (zero).
- If the two sets of characters compare equal, **strcmp()** returns 0 (zero).
- If the first set of characters (*s1*) compares less than the second (*s2*), **strcmp()** returns an **int** less than 0 (zero).

System issues

On UNIX, this function can be provided by the system library.

See Also

[memcmp\(\)](#) — [Compare two areas of memory](#)

[String-Handling Functions](#) on page 432 ([Comparing Strings](#))

Example

See the file *strcmp.c* in the *install_dir/docs/examples/* directory on your distribution.

strncpy()

Copy a number of characters from one string into another

Targets

strncpy() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */
char *strncpy(char *dest, const char *source, size_t n);
```

Description

strncpy() copies exactly *n* characters from the string *source* to the string *dest*.

- If a NUL character ('\0') terminates *source* before *n* characters have been copied, *dest* is NUL padded.
- If no NUL is encountered before *n* characters have been written into *dest*, the resulting character array is not NUL terminated.

strncpy() copies from left to right (low addresses to high addresses).

If *source* < *dest* < *source* + *n* (that is, if the source and destination areas overlap, with the destination area beginning within the source area), the behavior is undefined. In this situation, use [rstrncpy\(\)](#).

CAUTION **strncpy()** writes *n* bytes. After a call to **strncpy()**, it might not be safe to pass *dest* to functions that expect a string, as *dest* might not be terminated with a NUL.

Return Value

strncpy() returns *dest*.

Surprises

strncpy() copies the second argument to the first.

System issues

On UNIX, this function can be provided by the system library.

See Also

[memcpy\(\)](#) — [Copy from one place in memory to another](#)

[memset\(\)](#) — [Duplicate a character across an area of memory](#)

[rmemcpy\(\)](#) — [Copy from one place in memory to another](#)

[String-Handling Functions](#) on page 432 ([Copying Strings](#))

Example

See the file `strncpy.c` in the *install_dir/docs/examples/* directory on your distribution.

strnicmp() and _strnicmp()

Compare strings, ignoring case

Targets

strnicmp() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_strnicmp() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

strnicmp() is Non-ANSI; **_strnicmp()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>      /* Required */
int strnicmp(const char *s1, const char *s2, size_t count);
```

Description

strnicmp() and **_strnicmp()** compare up to *count* characters of the two strings *s1* and *s2*. If either string is shorter than *count* characters, the length of the shorter string is used in place of the *count*. The compare considers uppercase and lowercase alphabetic letters to be equal.

Return Value

This is the result of the comparison:

- If *s1* is less than *s2*, the function returns an **int** less than 0 (zero).
- If *s1* is greater than *s2*, the function returns an **int** greater than 0 (zero).
- If *s1* is equal to *s2*, the function returns 0 (zero).

System issues

On UNIX, this function can be provided by the system library.

See Also

[String-Handling Functions](#) on page 432 ([Comparing Strings](#))

Example

See the file *strnicmp.c* in the *install_dir/docs/examples/* directory on your distribution.

strnset() and _strnset()

Fill part or all of a string with any character

Targets

strnset() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_strnset() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

strnset() is Non-ANSI; **_strnset()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>    /* Required */
char *strnset(char * strptr, int c, size_t count);
```

Description

strnset() replaces the first *count* characters in the string pointed to by *strptr* with the character *c*, unless the end of the string is encountered first.

Return Value

`strnset()` returns *strptr*.

See Also

[memset\(\)](#) — Duplicate a character across an area of memory

[String-Handling Functions](#) on page 432 ([Filling Strings](#))

Example

See the file `strnset.c` in the *install_dir/docs/examples/* directory on your distribution.

strpbrk()

Find the first occurrence of any character of one string in another

Targets

`strpbrk()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */
char *strpbrk(const char *s1, const char *s2);
```

Description

`strpbrk()` finds the first occurrence of any character from the string *s2* in the string *s1*.

Return Value

`strpbrk()` returns a pointer to the character, if it is found. If not successful, it returns **NULL**.

System issues

On UNIX, this function can be provided by the system library.

See Also

[memchr\(\)](#) — Find a character in an area of memory

[String-Handling Functions](#) on page 432 ([Determining String Length](#) and [Finding Characters in Strings](#))

Example

See the file `strpbrk.c` in the *install_dir/docs/examples/* directory on your distribution.

strrchr()

Find the last occurrence of a character in a string

Targets

strrchr() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */  
char *strrchr(const char *s, int c);
```

Description

strrchr() finds the last occurrence of the character *c* (the least significant byte of the **int** *c*) in the string pointed to by *s*.

NUL is a valid value for *c* and results in **strrchr()** returning a pointer to the end of the string (pointing to the terminating NUL).

strrchr() takes *c* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to **strrchr()**, it is automatically coerced to type **int**.

Return Value

If the character indicated by *c* is found, **strrchr()** returns a pointer to the character. If not successful, it returns **NULL**.

System issues

On UNIX, this function can be provided by the system library.

See Also

[**memchr\(\)** — Find a character in an area of memory](#)

[String-Handling Functions](#) on page 432 ([Determining String Length](#) and [Finding Characters in Strings](#))

Example

See the file *strrchr.c* in the *install_dir/docs/examples/* directory on your distribution.

strrev() and _strrev()

Reverse the order of characters in a string

Targets

strrev() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_strrev() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

strrev() is Non-ANSI; _strrev() is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>          /* Required */
char *strrev(char *strptr);
char *_strrev(char *strptr);
```

Description

strrev() reverses the order of the characters in the string pointed to by *strptr*. The first character is exchanged with the last character, then the next character is exchanged with the next to last, and so on, until all characters have been exchanged.

Return Value

strrev() returns *strptr*.

See Also

[swab\(\) and _swab\(\)](#) — Swap bytes in a word

[String-Handling Functions](#) on page 432 ([Copying Strings](#) and [Filling Strings](#))

Example

See the file `strrev.c` in the *install_dir/docs/examples/* directory on your distribution.

strset() and _strset()

Fill a string with any character

Targets

strset() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_strset() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

strset() is Non-ANSI; _strset() is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h>          /* Required */
char *strset(char *strptr, int c);
char *_strset(char *strptr, int c);
```

Description

strset() replaces every character in the string pointed to by *strptr* with the character *c*, except for the string's terminating NUL character ('\0').

Return Value

strset() returns *strptr*.

See Also

[String-Handling Functions](#) on page 432 ([Filling Strings](#))

Example

See the file *strset.c* in the *install_dir/docs/examples/* directory on your distribution.

strspn()

Determine the length of the prefix of a string composed entirely of characters from another string

Targets

strspn() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */
size_t strspn(const char *s1, const char *s2);
```

Description

strspn() finds the first substring equal to *s2* in *s1*. NUL is not used in the comparison process.

Return Value

strspn() returns the length of that segment of the string *s1* that begins at the beginning of the string *s1* and is made up entirely of characters that occur in the string *s2*. The terminating NUL character ('\0') is not considered part of *s2* for this purpose.

If the string *s1* begins with a character not in *s2*, there is no substring, and **strspn()** returns 0 (zero).

System issues

On UNIX, this function can be provided by the system library.

See Also

[memchr\(\)](#) — [Find a character in an area of memory](#)

[String-Handling Functions](#) on page 432 ([Finding Characters in Strings](#) and [Determining String Length](#))

strrstr()

Find the first occurrence of one string within another

Targets

strrstr() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>          /* Required */
char *strrstr(const char *s1, const char *s2);
```

Description

strrstr() locates the first occurrence of *s2* (not including the terminating NUL) in *s1*.

Return Value

strrstr() returns a pointer to the first occurrence of *s2* in *s1*. If no occurrence is found, **strrstr()** returns NULL. If *s2* has zero length, **strrstr()** returns *s1*.

See Also

[String-Handling Functions](#) on page 432 ([Finding Characters in Strings](#))

Example

See the file *strrstr.c* in the *install_dir/docs/examples/* directory on your distribution.

_strtime()

Convert current time of day to a string

Targets

_strtime() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <time.h>          /* Required */
char *_strtime(char *sptr);
```

Description

_strtime() obtains the current time of day, converts it to character format, and places it in the string pointed to by *sptr*.

The time is formatted as *hh:mm:ss*, where *hh* is hours (24-hour format), *mm* is minutes, and *ss* is seconds. Each part of the time is two digits, and the string that *sprt* points to must be at least nine bytes long. The function places a terminating NUL character ('\0') at the end of the time string.

Return Value

`_strtime()` returns *sprt*.

See Also

[String-Handling Functions](#) on page 432 ([Converting Other Data to Strings](#))

[Time-Related Functions](#) on page 434

Example

See the file `strtime.c` in the *install_dir/docs/examples/* directory on your distribution.

strtod()

Convert a string to a **double**

Targets

`strtod()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */
double strtod(const char *nptr, char **endptr);
```

Description

`strtod()` converts a prefix of the string *nptr* to a floating-point number. A pointer into *nptr*, pointing at the place where the conversion left off, is placed in the pointer referenced by *endptr*.

`strtod()` recognizes strings of the form:

Wspace? Sign? Dgts('.'Dgts)?((e'|E')Sign? Dgts)?

In this case, Wspace represents Whitespace and Dgts represents Digits (decimal digits), as explained in [Regular Expressions](#) on page 18.

The first character that is not in this sequence ends the conversion. If *endptr* is not **NULL**, `strtod()` stores a pointer to that character in the object to which *endptr* points.

Return Value

`strtod()` returns the value of the floating-point number resulting from the conversion.

- If the correct value is outside the range of representable values, `strtod()` returns plus or minus [HUGE_VAL](#) and sets global variable `errno` to [ERANGE](#) ([Mathematical range error](#)).
- If the correct value causes underflow, `strtod()` returns 0 (zero) and sets global variable `errno` to [ERANGE](#).

If the string is empty, no Dgts are found in the sequence, or the only Dgts found follow the 'e' or 'E', `strtod()` returns 0 (zero).

See Also

[Conversion Functions](#) on page 425

[String-Handling Functions](#) on page 432 ([Converting, Transforming, or Formatting Strings](#))

Global variable `errno` — [An int used to record error numbers](#)

Example

See the file `strtod.c` in the `install_dir/docs/examples/` directory on your distribution.

strtok()

Divide a string into tokens

Targets

`strtok()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <string.h>           /* Required */
char *strtok(char *tokens, const char *delims);
```

Description

`strtok()` treats the string *tokens* as a series of tokens. The tokens are made up of characters not contained in the string *delims*, and are separated by one or more characters that are in *delims*. (That is, *delims* defines the token delimiters.)

The first call to `strtok()` finds the first token, writes a NUL character ('\0') into *tokens* at the end of the token, and returns a pointer to the first character of the token; that is, `strtok()` creates and returns a substring consisting of the token.

Each subsequent call to `strtok(NULL,delims)` causes a pointer to the next NUL-terminated token to be returned in the same manner. (Note that the first argument is the null pointer; this causes `strtok()` to search the previous token string.)

The token-separator string *delims* can change from call to call.

CAUTION `strtok()` writes on *tokens*.

Return Value

`strtok()` returns a pointer to the first token in *tokens* on the first call to `strtok()`. On each successive call, `strtok()` returns a pointer to the next token. If `strtok()` is called and no token is found, `strtok()` returns `NULL`.

See Also[Conversion Functions](#) on page 425[String-Handling Functions](#) on page 432 ([Converting, Transforming, or Formatting Strings](#))**Example**See the file `strtok.c` in the `install_dir/docs/examples/` directory on your distribution.

strtol()

Convert a string to a **long****Targets****strtol()** works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h> /* Required */
long strtol(const char *nptr, char **endptr, int base);
```

Description**strtol()** converts a prefix of the string *nptr* to a **long int**.**strtol()** recognizes strings of the form:

Wspace? Sign? ('0' ('x' | 'X'))? Digits

In this case, Wspace represents Whitespace, as explained in [Regular Expressions](#) on page 18, and Digits represents all possible digits ('0' — '9' and 'a' — 'z' or 'A' — 'Z'). '0x' or '0X' can appear in the string only if *base* is 16 or 0 (zero).

The first character that is not in this sequence ends the conversion.

If *endptr* is not **NULL**, a pointer into *nptr*, pointing to the place where the conversion left off, is placed in the pointer referenced by *endptr*.**Determining Which Base to Use for Conversion**If *base* is 0 (zero), *nptr* itself determines which of three bases (16, 8, or 10) will be used for conversion. Following the sign (if any):

- A leading 0x or 0X indicates a hexadecimal number: base 16 is used.
- A leading 0 (zero) indicates an octal number: base 8 is used.
- In all other cases, base 10 is used.

If *base* is between 2 and 36, inclusive, it is used as the base for conversion.

- If *base* > 10, alphabetic characters are available for use as Digits (as for hexadecimal numbers), carried to the logical limit. For example, in base 21 to base 36, K or k denotes decimal 20.

- If *base* is 16, 0x or 0X can prefix the digit string.

If *base* == 1, or if *base* > 36, it is invalid and **strtol()** returns 0 (zero).

Return Value

strtol() returns the prefix of the string *nptr* converted to a **long int**.

If the number is too large to be expressed as a **long int**, **strtol()** sets global variable **errno** to **ERANGE** ([Mathematical range error](#)) and returns **LONG_MAX** or **LONG_MIN**, depending on the sign of the input.

If the string is empty, the *base* is invalid, or no Digits are found in the sequence, **strtol()** returns 0 (zero).

See Also

[**sprintf\(\)** — Print to a string](#)

[**Conversion Functions** on page 425](#)

[**String-Handling Functions** on page 432 \(\[Converting, Transforming, or Formatting Strings\]\(#\)\)](#)

Constant [**LONG *** — Minimum and maximum values for signed long types](#)

Global variable [**errno** — An int used to record error numbers](#)

Example

See the file `strtoul.c` in the *install_dir/docs/examples/* directory on your distribution.

strtoul()

Convert a string to an **unsigned long**

Targets

strtoul() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */
unsigned long strtoul(const char *nptr, char **endptr, int base);
```

Description

strtoul() converts a prefix of the string *nptr* to an **unsigned long int**. **strtoul()** recognizes strings of the form:

Whitespace? ('0' ('x' | 'X'))? Digits

The first character that is not in the described sequence ends the conversion. (See [Regular Expressions](#) on page 18 for an explanation of this notation.)

If *endptr* is not **NULL**, a pointer into *nptr*, pointing to the place where the conversion left off, is placed in the pointer referenced by *endptr*.

0x or 0X can appear in the string only if *base* is 16 or 0 (zero).

If *base* is 0 (zero), *nptr* itself determines which of three bases (16, 8, or 10) will be used for conversion.

For a description of how the base is determined, see the section [Determining Which Base to Use for Conversion](#) in the description of [strtol\(\)](#).

Return Value

[strtoul\(\)](#) returns the prefix of the string *nptr* converted to a **unsigned long int**.

If the number is too large to be expressed as an **unsigned long int**, [strtoul\(\)](#) sets global variable *errno* to [ERANGE](#) ([Mathematical range error](#)) and returns **ULONG_MAX**.

If the string *nptr* is empty or no Digits are found in the sequence, [strtoul\(\)](#) returns 0 (zero).

See Also

[sprintf\(\)](#) — Print to a string

[Conversion Functions](#) on page 425

[String-Handling Functions](#) on page 432 ([Converting, Transforming, or Formatting Strings](#))

Constants [U*_MAX](#) — [Maximum values for unsigned types](#)

Global variable [errno](#) — [An int used to record error numbers](#)

strupr() and _strupr()

Make string all uppercase

Targets

[strupr\(\)](#) works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

[_strupr\(\)](#) works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

[strupr\(\)](#) is Non-ANSI; [_strupr\(\)](#) is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <string.h> /* Required */
char *strupr(char *strptr);
```

Description

[strupr\(\)](#) converts all lowercase alphabetic letters in the string pointed to by *strptr* into their equivalent uppercase letters. Other characters are not modified.

System issues

On UNIX, this function can be provided by the system library.

Return Value

`strupr()` returns `strptr`.

See Also

[String-Handling Functions](#) on page 432 ([Converting, Transforming, or Formatting Strings](#))

Example

See the file `strupr.c` in the `install_dir/docs/examples/` directory on your distribution.

strxfrm()

Transform string to locale-independent form

Targets

`strxfrm()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <string.h>           /* Required */
size_t strxfrm(char *s1, char *s2, size_t n);
```

Description

`strxfrm()` transforms the string `s2` into the string `s1` in such a way that if `strcmp()` is applied to two such transformed strings, the result will be the same as if `strcmp()` were applied to the same two original strings.

MetaWare C supports only the standard locale, so no transformation is done.

Return Value

`strxfrm()` returns the length L of the transformed result R ; that is, $L=\text{strlen}(R)$.

- If n is greater than L , this transformed result is placed into `s1`.
- If n is less than or equal to L , the contents of `s1` are indeterminate.
- If `s1` and `s2` overlap, the behavior of `strxfrm()` is undefined.

Surprises

`strxfrm()` transforms the second argument to the first.

See Also

[String-Handling Functions](#) on page 432 ([Comparing Strings](#) and [Converting, Transforming, or Formatting Strings](#))

swab() and _swab()

Swap bytes in a word

Targets

swab() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_swab works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

swab() is Non-ANSI; **_swab()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h> /* Required */
void swab(char *source, char *target, int count);
```

Description

swab() copies *count* bytes from string *source* into string *target*. As each even/odd pair of bytes is copied, their order is reversed; that is:

- *source*[0] is copied to *target*[1], and *source*[1] is copied to *target*[0]
- *source*[2] is copied to *target*[3], and *source*[3] is copied to *target*[2]

and similarly for each even/odd pair of bytes in the data areas.

The **swab()** function has frequent application for machines with differing byte/word ordering.

See Also

[String-Handling Functions](#) on page 432 ([Copying Strings](#) and [Converting, Transforming, or Formatting Strings](#))

Example

See the file *swab.c* in the *install_dir/docs/examples/* directory on your distribution.

system()

Pass a command to the operating system

Targets

system() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdlib.h>
int system(const char *string);
```

Description

system() passes *string* to be executed as a command by the host environment's command processor. Use **system()** to check if the command processor exists by setting *string* to **NULL**.

Return Value

If successful, **system()** invokes **_spawnl()** on the command processor and reflects the return value from **_spawnl()**.

If it is unable to find the command processor, **system()** returns -1 and sets global variable **errno** to [ENOENT \(File or directory not found; pathname element does not exist\)](#). If *string* is **NULL** or empty, **system()** returns 1 (one).

System issues

On DOS, executing processes from within your program requires enough conventional memory (memory below 640K) to be available for running the executable. If insufficient conventional memory is available, the results are undefined.

Conventional memory can be controlled with Phar Lap 386|DOS-Extender options **-MAXREAL** and **-MINREAL**. See the [Phar Lap 386|DOS-Extender Reference Manual](#).

On UNIX, **system()** is not supported as a function call. It is provided by the operating system on all platforms.

On embedded systems, calling **system()** might just return an error.

See Also

[**spawn*\(\)** and **_spawn*\(\)** — Execute a child process](#)

Example

See the file **system.c** in the *install_dir/docs/examples/* directory on your distribution.

tan()

Tangent

Targets

tan() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double tan(double x);
```

Description

tan calculates the tangent of x . A large-magnitude argument might yield a result with little or no significance. **tan** might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

tan returns the tangent of x , where x is in radians.

See Also

[Transcendental Math Functions](#) on page 424

[**atan\(\)** — Arc tangent](#)

[**atan2\(\)** — Arc tangent of the angle defined by a point](#)

[**atanh\(\) and _atanh\(\)** — Hyperbolic arc tangent](#)

[**cos\(\)** — Cosine](#)

[**sin\(\)** — Sine](#)

[**tanf\(\)** — Tangent using single-precision floating point](#)

[**tanh\(\)** — Hyperbolic tangent](#)

tanf()

Tangent using single-precision floating point

Targets

tanf works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
float tanf(float x);
```

Description

tanf calculates the tangent of x . A large-magnitude argument might yield a result with little or no significance. **tanf** might be inlined when the toggle `Recognize_library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

tanf returns the tangent of x , where x is in radians.

See Also

[Transcendental Math Functions](#) on page 424

[**atan\(\)** — Arc tangent](#)

[**atan2\(\)** — Arc tangent of the angle defined by a point](#)

[atan2f\(\)](#) — Arc tangent of the angle defined by a point, using single-precision floating point

[atanf\(\)](#) — Arc tangent using single-precision floating point

[atanh\(\) and _atanh\(\)](#) — Hyperbolic arc tangent

[cos\(\)](#) — Cosine

[cosf\(\)](#) — Cosine using single-precision floating point

[sin\(\)](#) — Sine

[sinf\(\)](#) — Sine using single-precision floating point

[tan\(\)](#) — Tangent

[tanh\(\)](#) — Hyperbolic tangent

tanh()

Hyperbolic tangent

Targets

tanh works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */
double tanh(double x);
```

Description

tanh calculates the hyperbolic tangent of *x*.

tanh might be inlined when the toggle **Recognize_library** is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

tanh returns the hyperbolic tangent of *x*.

If the magnitude of *x* is too large, such that **sinh(x)** or **cosh(x)** cannot be represented, **tanh** sets global variable **errno** to [ERANGE](#) ([Mathematical range error](#)) and returns [HUGE_VAL](#) with the same sign as *x*.

See Also

[Transcendental Math Functions](#) on page 424

[atan\(\)](#) — Arc tangent

[atan2\(\)](#) — Arc tangent of the angle defined by a point

[atanh\(\) and _atanh\(\)](#) — Hyperbolic arc tangent

[coshf\(\)](#) — Hyperbolic cosine using single-precision floating point

[sinh\(\)](#) — Hyperbolic sine

[tan\(\)](#) — Tangent

[tanhf\(\)](#) — Hyperbolic tangent using single-precision floating point

Global variable [errno](#) — An int used to record error numbers

tanhf()

Hyperbolic tangent using single-precision floating point

Targets

tanhf works on the following target platforms:

EMB	UNIX	DOS	NT
x	—	—	—

ANSI

ANSI C99 compliant

Reentrancy

Reentrant

Synopsis

```
#include <math.h>           /* Required */  
float tanhf(float x);
```

Description

tanhf calculates the hyperbolic tangent of x .

tanhf might be inlined when the toggle `Recognize_Library` is On. Refer to the *MetaWare C/C++ Programmer's Guide* for details.

Return Value

tanhf returns the hyperbolic tangent of x .

If the magnitude of x is too large, such that **sinh**(x) or **cosh**(x) cannot be represented, **tanhf** sets global variable **errno** to [ERANGE](#) ([Mathematical range error](#)) and returns [HUGE_VAL](#) with the same sign as x .

See Also

[Transcendental Math Functions](#) on page 424

[atan\(\)](#) — Arc tangent

[atan2\(\)](#) — Arc tangent of the angle defined by a point

[atan2f\(\)](#) — Arc tangent of the angle defined by a point, using single-precision floating point

[atanf\(\)](#) — Arc tangent using single-precision floating point

[atanh\(\) and _atanh\(\)](#) — Hyperbolic arc tangent

[cosh\(\)](#) — Hyperbolic cosine

[coshf\(\)](#) — Hyperbolic cosine using single-precision floating point

[sinh\(\)](#) — [Hyperbolic sine](#)

[sinhf\(\)](#) — [Hyperbolic sine using single-precision floating point](#)

[tan\(\)](#) — [Tangent](#)

[tanf\(\)](#) — [Tangent using single-precision floating point](#)

[tanh\(\)](#) — [Hyperbolic tangent](#)

Global variable [errno](#) — [An int used to record error numbers](#)

tell() and _tell()

Report the current position in a file

Targets

tell() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_tell() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

tell is Non-ANSI; **_tell** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h> /* Required */
long tell(int file_handle);
long _tell(int file_handle);
```

Description

tell reports the current position in a file. The current position is the place where the next byte will be read from or written to. It is measured as the number of bytes from the start of the file.

Return Value

If *file_handle* is a valid file handle, **tell** returns the current position in the file, measured as the number of bytes from the beginning of the file. If *file_handle* is invalid, **tell** returns -1 and sets global variable [errno](#) to [EBADF](#) (“Bad” (invalid or inaccessible) file handle).

If *file_handle* refers to a character device, the file position and the return value are undefined.

See Also

[File-Related Functions](#) on page 429

Example

See the file *tell.c* in the *install_dir/docs/examples/* directory on your distribution.

_tempnam()

Generate a temporary file name in a directory

Targets

_tempnam works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h> /* Required */
char *_tempnam(char *dir, char *prefix);
```

Description

_tempnam creates a unique file name for a file in a directory.

- The target directory is chosen by evaluating the following conditions in the order given here:
 1. If environment variable TMP (for DOS or Windows targets) or TMPDIR (for UNIX targets) is set to an existing directory, **_tempnam** chooses the directory specified by TMP (or TMPDIR).
 2. If environment variable TMP (or TMPDIR) is not set to an existing directory, **_tempnam** chooses the directory specified by the *dir* argument to **_tempnam**.
 3. If *dir* is **NULL**, or if *dir* names a non-existent directory, **_tempnam** chooses the directory specified by **P_tmpdir** in **stdio.h**.
 4. If **P_tmpdir** does not exist, **_tempnam** chooses the current working directory.
- *prefix* specifies the first part of the unique file name. *prefix* cannot contain a period and must contain one to five characters.

_tempnam concatenates the directory name, *prefix*, and six unique characters to produce the file name. Space for the file name is allocated using **malloc** and must be deallocated with a call to **free** when no longer needed.

NOTE You must explicitly delete any temporary file that you create with **_tempnam**; the file is not deleted automatically.

Return Value

If **_tempnam** can open one of the previously listed target directories, it returns a pointer to *dir*, which contains the unique file name. Use this pointer as an argument to **free** to deallocate the space for *dir*.

If **_tempnam** cannot open any of the target directories, it returns **NULL**.

See Also

[**free\(\)** — Release storage allocated by **calloc\(\)**, **malloc\(\)**, or **realloc\(\)**](#)

[**malloc\(\)** — Dynamically allocate uninitialized storage](#)

[**tmpnam\(\)**](#) — Generate a string to be used as a temporary file name

Example

See the file `tmpnam.c` in the `install_dir/docs/examples/` directory on your distribution.

time()

Get the current time and date

Targets

time works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <time.h> /* Required */
time_t time(time_t *timer);
```

Description

time gets the current system time and date. *timer* points to a **time_t** type in which **time** stores the time and date data. Arithmetic type **time_t**, defined in `time.h`, represents the time and date.

Return Value

time returns the current time and date, to the best of the system's knowledge and approximation, in a form that is understood by the other functions provided in `time.h`. If *timer* is not **NULL**, the return value is also stored in the object to which it points.

See Also

[Time-Related Functions](#) on page 434

[**CLOCKS_PER_SEC**](#) — Clock ticks converted to seconds

Type [**clock_t**](#) — Representation of elapsed processor time

Type [**time_t**](#) — Representation of a date and time

Example

See the file `asctime.c` in the `install_dir/docs/examples/` directory on your distribution.

tmpfile() and _tmpfile()

Create a temporary file

Targets

tmpfile() and **_tmpfile()** work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`tmpfile()` is ANSI compliant; `_tmpfile()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>           /* Required */
FILE *tmpfile(void);
FILE *_tmpfile(void);
```

Description

`tmpfile()` and `_tmpfile()` create a temporary file. The file is opened as a binary file for update and is deleted when it is closed or when the program terminates.

NOTE If the program terminates abnormally, the file might not be deleted.

Return Value

`tmpfile()` and `_tmpfile()` return a pointer to the controlling `FILE` variable of the temporary file. If the file cannot be created, `tmpfile()` and `_tmpfile()` return `NULL`.

See Also

[File-Related Functions](#) on page 429

`typedef struct FILE — Information for controlling a stream`

tmpnam()

Generate a string to be used as a temporary file name

Targets

`tmpnam()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
char *tmpnam(char *s);
```

Description

`tmpnam()` generates a string that can be used as a temporary file name. `tmpnam()` generates a different file name each time it is called (up to `TMP_MAX` times). Other than being given names that are unlikely to clash with file names generated by users, there is nothing temporary about the files — they must still be opened, flushed, removed, and so forth, by standard library calls.

CAUTION If *s* is not **NULL**, **tmpnam()** writes on whatever it points to, up to the number of bytes specified by **L_tmpnam**.

Return Value

If *s* is not **NULL**, **tmpnam()** writes on and returns the string *s*. If *s* is **NULL**, **tmpnam()** writes the file name in an internal **static** string and returns the file name. Subsequent calls to **tmpnam()** overwrite that string.

See Also

[File-Related Functions](#) on page 429

Constant [**L_tmpnam**](#) — [Size of a temporary file name](#)

Constant [**TMP_MAX**](#) — [Minimum number of unique file names generated by tmpnam\(\)](#)

Example

See the file `remove.c` in the *install_dir/docs/examples/* directory on your distribution.

tolower()

Convert to lowercase

Targets

tolower() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>
int tolower(int c);
```

Description

tolower() converts an uppercase letter to a lowercase letter.

Return Value

If *c* is an uppercase letter, **tolower()** returns the corresponding lowercase letter. If *c* is not an uppercase letter, **tolower()** returns *c*.

See Also

[**tolower\(\)**](#) — [Convert to lowercase without testing for uppercase](#)

[**toupper\(\)**](#) — [Convert to uppercase](#)

[**toupper\(\)**](#) — [Convert to uppercase without testing for lowercase](#)

Example

See the file `tolower.c` in the *install_dir/docs/examples/* directory on your distribution.

tolower()

Convert to lowercase without testing for uppercase

Targets

tolower() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h> /* Required for macro */  
int _tolower(int c);
```

Description

tolower() is a version of **tolower()** that can be used when *c* is known to be an uppercase character. You must always test the argument *c* with **isupper()**. If *c* is not an uppercase letter, the behavior of tolower() is undefined.

NOTE tolower() is provided both as a macro and as a function. (See [Using Macros](#) on page 24.)

Return Value

tolower() returns the lowercase letter corresponding to *c*, if *c* is an uppercase letter.

See Also

[isupper\(\) — Test for uppercase alphabetic character](#)

[tolower\(\) — Convert to lowercase](#)

[toupper\(\) — Convert to uppercase](#)

[toupper\(\) — Convert to uppercase without testing for lowercase](#)

Example

See the file *tolower2.c* in the *install_dir/docs/examples/* directory on your distribution.

toupper()

Convert to uppercase

Targets

toupper() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>
int toupper(int c);
```

Description

toupper() converts a lowercase letter to an uppercase letter.

Return Value

If *c* is a lowercase letter, **toupper()** returns the corresponding uppercase letter. If *c* is not a lowercase letter, **toupper()** returns *c*.

See Also

[tolower\(\)](#) — [Convert to lowercase](#)

[tolower\(\)](#) — [Convert to lowercase without testing for uppercase](#)

[toupper\(\)](#) — [Convert to uppercase without testing for lowercase](#)

Example

See the file *toupper.c* in the *install_dir/docs/examples/* directory on your distribution.

_toupper()

Convert to uppercase without testing for lowercase

Targets

_toupper() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <ctype.h>    /* Required for macro */
int _toupper(int c);
```

Description

_toupper() is a version of **toupper()** that can be used when *c* is known to be a lowercase character. You must always test the argument *c* with **islower()**. If *c* is not a lowercase letter, the behavior of **_toupper()** is undefined.

NOTE **_toupper()** is provided both as a macro and as a function. (See [Using Macros](#) on page 24.)

Return Value

`_toupper()` returns the uppercase letter corresponding to *c*, if *c* is a lowercase letter.

See Also

[`islower\(\)`](#) — [Test for lowercase alphabetic character](#)

[`tolower\(\)`](#) — [Convert to lowercase](#)

[`tolower\(\)`](#) — [Convert to lowercase without testing for uppercase](#)

[`toupper\(\)`](#) — [Convert to uppercase](#)

Example

See the file `toupper2.c` in the *install_dir/docs/examples/* directory on your distribution.

tzset() and _tzset()

Set local time-zone values

Targets

`tzset()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_tzset()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`tzset()` is Non-ANSI; `_tzset()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <time.h>
void tzset(void);
void _tzset(void);
```

Description

`_tzset()` uses the environment variable TZ to determine the local time zone. The default value for TZ is PST8PDT.

Environment Variable TZ

The TZ setting is made by an operating-system command, and must be of this form:

set TZ=SSSHHDDD

The components of the TZ string are defined as follows:

SSS

Name of the local time zone; always a three-character constant. SSS is placed in `_tzname[0]` in the global variable `char *_tzname`. (Default = PST.)

HH

Number of hours west of Coordinated Universal Time (UTC). If east of UTC, a negative number would be used; for example, -5. (Default = None.)

DDD

Name of the local time zone when Daylight Saving Time is in effect. *DDD* is placed in `_tzname[1]` in the global variable `char *_tzname`. (Default = PDT.)

The number of hours from UTC is converted to seconds and placed in the global variable `timezone`. The global variable `_daylight` is set to 1 (one) if PDT is present, and to 0 (zero) if PDT is not present.

See Also

[Time-Related Functions](#) on page 434

Global variable `timezone` — [Time zone; hours from UTC, expressed in seconds](#)

Global variable `_daylight` — [Pacific Daylight Time](#)

Example

See the file `tzset.c` in the `install_dir/docs/examples/` directory on your distribution.

_udiv0()

Replaceable handler for unsigned integer division by zero (ARC only)

Targets

`_udiv0()` works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>
extern unsigned _udiv0(unsigned dividend);
```

Description

The run-time library for ARC targets calls `_udiv0()` when unsigned integer division detects division by zero. The default version of `_udiv0()` returns the maximum unsigned integer value. You can supply your own version to implement the behavior you desire.

Note If you replace `_udiv0()` with your own functionality, you must also replace `_div0()`.

Return Value

`_udiv0()` returns the maximum unsigned integer value.

See Also

[_div0\(\)](#) — Replaceable handler for integer division by zero (ARC only)

ultoa() and _ultoa()

Convert an **unsigned long** to an ASCII string

Targets

ultoa() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_ultoa() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ultoa() is Non-ANSI; **_ultoa()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h>          /* Required */
char *ultoa(unsigned long val, char *target, int radix);
char *_ultoa(unsigned long val, char *target, int radix);
```

Description

ultoa() and **_ultoa()** convert any **long** integer *val* into an ASCII string *target*. The conversion is performed according to the number system specified by *radix*.

radix can be any number between 2 and 36. When *radix* is 2, the maximum size for *target* is 33 bytes, including the terminating null byte. When *radix* is 10, the maximum *target* is 11 bytes, consisting of 10 digits and the final null byte.

Return Value

ultoa() and **_ultoa()** return *target*.

See Also

[Conversion Functions](#) on page 425

[**utoa\(\)** and **_utoa\(\)** — Convert an unsigned int to an ASCII string](#)

Example

See the file `ultoa.c` in the *install_dir/docs/examples/* directory on your distribution.

umask() and _umask()

Set global file permission mask

Targets

umask() and **_umask()** work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

umask() is Non-ANSI; **_umask()** is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
int umask(int permits);
```

Description

umask() and **_umask()** set the global file-permission mask to the value of *permits*. This mask is used by the **open()**, **creat()**, and **sopen()** functions. Any bit set in the mask denies the corresponding privilege. The actual file permissions are not set until the file is closed for the first time.

The two values allowed in the mask are **_S_IWRITE** and **_S_IREAD**, as defined in **stat.h**. They can also be combined with a logical OR operator (**|**) as (**_S_IWRITE | _S_IREAD**).

When **_S_IWRITE** is set with the **umask()** or **_umask()** function, all files created with the **creat()**, **open()**, and **sopen()** functions after that point will have write access denied after they are closed.

Return Value

umask() and **_umask()** return the previous value of the file-permission mask.

System issues

On DOS and Windows NT, all files are readable (it is not possible to give write-only permission), so the read permission is included for compatibility with other systems, but is ignored here.

See Also

[File-Related Functions](#) on page 429

Example

See the file **umask.c** in the *install_dir/docs/examples/* directory on your distribution.

ungetc()

Push a character back into an input stream

Targets

ungetc() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
int ungetc(int c, FILE *stream);
```

Description

`ungetc()` pushes the character *c* (the least significant byte of the `int c`) back into the input stream *stream*. The character is returned by the next read on *stream*, assuming that none of [fseek\(\)](#), [rewind\(\)](#), or [fsetpos\(\)](#) is called first. `fseek()`, `fsetpos()`, and `rewind()` erase all memory of pushed-back characters.

`ungetc()` does not change the external storage associated with *stream*.

Return Value

If successful, `ungetc()` returns the character pushed back into the stream. If it could not push the character back into the stream, `ungetc()` returns `EOF`.

Surprises

`ungetc()` takes *c* as an `int` rather than a `char` argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type `char` is passed to `ungetc()`, it is automatically coerced to type `int`.

See Also

[I/O Functions](#) on page 427

Constant `EOF` — [End-of-file indicator](#)

`typedef struct FILE` — [Information for controlling a stream](#)

Example

See the file `ungetc.c` in the *install_dir/docs/examples/* directory on your distribution.

ungetch() and _ungetch()

Push a character back into the input buffer

Targets

`ungetch()` and `_ungetch()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`ungetch()` is Non-ANSI; `_ungetch()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <conio.h>
int ungetch(int c);
int _ungetch(int c);
```

Description

`ungetch()` and `_ungetch()` force the character *c* back into the input buffer. The next call to `getch()` or `getche()` returns the character *c*, before any other pending input characters. Only one character can be forced back into input with this function. The character *c* cannot be the EOF code.

Return Value

`ungetch()` and `_ungetch()` return the character *c*.

See Also[I/O Functions](#) on page 427**Example**See the file `ungetch.c` in the `install_dir/docs/examples/` directory on your distribution.

unlink() and _unlink()

Delete a file

Targets

unlink() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

_unlink() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	X	X

ANSI

unlink() is Non-ANSI; _unlink() is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>
int unlink(const char *pathname);
int _unlink(const char *pathname);
```

Descriptionunlink() and _unlink() remove the file named by the string *pathname*. For ANSI compliance use the `remove()` function instead.

CAUTION If `unlink()` is called on an open file, the behavior is implementation-defined and the return value might be meaningless.

System issues

On UNIX and embedded systems, this function can be provided by the system library or some third-party library.

Return ValueIf the operation is successful, `unlink()` and `_unlink()` return 0 (zero). If the operation fails, `unlink()` and `_unlink()` set `errno` such that a call to `perror()` will result in an error message describing the reason for the failure, and return the (non-zero) value of `errno`.**See Also**[fclose\(\)](#) — Close a file[perror\(\)](#) — Print an error message[remove\(\)](#) — Delete a file

[rename\(\)](#) — Change the name of a file

Global variable [errno](#) — An int used to record error numbers

Example

See the file `unlink.c` in the `install_dir/docs/examples/` directory on your distribution.

utime() and _utime()

Update a file's date/time stamp

Targets

`utime()` and `_utime()` work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

`utime()` is Non-ANSI; `_utime()` is Extra-ANSI.

Reentrancy

Both functions are Not-Reentrant.

Synopsis

```
#include <sys/types.h>          /* Required */  
#include <sys/utime.h>          /* Required */  
int utime(char *pathname, struct utimbuf *newtime);
```

Description

`utime()` and `_utime()` change the date and time of the file specified by `pathname` to the date and time given in the `modtime` field of the `newtime` structure. These functions open the file (write access and a file handle must be available), set the date and time of the file, and close the file.

If `newtime` points to `NULL`, `utime()` and `_utime()` obtain the current date and time from the operating system and apply them to the file.

Return Value

If the operation is successful, `utime()` and `_utime()` return 0 (zero). If the operation is not successful, these functions return -1 and set global variable `errno` to one of the following values:

[EACCES](#) — Access error; permission denied

[EINVAL](#) — Invalid argument given

[EMFILE](#) — Too many open file handles

[ENOENT](#) — File or directory not found; pathname element does not exist

See Also

[Time-Related Functions](#) on page 434

[stat\(\)](#) and [_stat\(\)](#) — Get information about a file or directory

[struct utimbuf](#) — Information about file access and modification times

Example

See the file `utime.c` in the `install_dir/docs/examples/` directory on your distribution.

utoa() and _utoa()

Convert an **unsigned int** to an ASCII string

Targets

utoa() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

_utoa() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

utoa() is Non-ANSI; **_utoa()** is Extra-ANSI.

Reentrancy

Both functions are Reentrant.

Synopsis

```
#include <stdlib.h>          /* Required */
char *utoa(unsigned int val, char *buffer, int radix);
char *_utoa(unsigned int val, char *buffer, int radix);
```

Description

utoa() and **_utoa()** convert *val* to an ASCII string in base *radix* and place the result in *buffer*.

radix can be any number between 2 and 36. When *radix* is 2, the maximum size for *target* is 33 bytes, including the terminating null byte. When *radix* is 10, the maximum *target* is 11 bytes, consisting of 10 digits and the final null byte.

Return Value

utoa() and **_utoa()** return a pointer to the result of the conversion stored in *buffer*.

See Also

[Conversion Functions](#) on page 425

[**ultoa\(\) and _ultoa\(\)** — Convert an unsigned long to an ASCII string](#)

va_arg()

Get the next argument in a variable series of arguments

Targets

va_arg() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdarg.h>           /* Required */  
void va_arg(va_list ap, typeSpecifier);
```

Description

va_arg() gets the value of the next (as yet unreferenced by **va_arg()**) argument of a function whose right-most named argument was previously passed to **va_start()**.

va_start() must be called before the first call to **va_arg()**.

NOTE **va_arg()** is provided as a macro only.

typeSpecifier must be a type specifier such that a pointer to an object of that type can be obtained by appending an asterisk (*) to *typeSpecifier*. The type specifier should agree with the type of the corresponding argument (as widened by default function-call conversion). *typeSpecifier* is evaluated more than once. This should be harmless, because a type specifier should not have side effects

CAUTION **va_arg()** assumes that *ap* has been initialized by **va_start()**. Calling **va_arg()** when **va_start()** has not been called correctly results in an indeterminate return value.

Return Value

The first call to **va_arg()** returns the value of the argument following the one passed to **va_start()**. Subsequent calls to **va_arg()** return the values of corresponding subsequent arguments. **va_arg()** has no knowledge of the number of actual arguments available for referencing. Calling **va_arg()** after all arguments have been referenced (by **va_arg()**) results in an indeterminate return value.

See Also

[va_end\(\)](#) — [Terminate va_list\(\) processing](#)

[va_start\(\)](#) — [Initialize an object of type va_list](#)

Type [va_list](#) — [Information needed by var-arg macros](#)

Example

See the file `vaarg.c` in the *install_dir/docs/examples/* directory on your distribution.

va_end()

Terminate **va_list()** processing

Targets

va_end() works on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdarg.h>          /* Required */
void va_end(va_list ap);
```

Description

va_end() cleans up the processing of unnamed arguments so that a normal return can occur from a function using the var-arg (variable-argument) macros. **va_end()** must be called after all arguments have been accessed. On some systems **va_end()** expands to nothing so it has no cost; for portability it is best to always use **va_end()**.

NOTE **va_end()** is provided as a macro only.

See Also

[va_arg\(\)](#) — Get the next argument in a variable series of arguments

[va_start\(\)](#) — Initialize an object of type **va_list**

Type [va_list](#) — Information needed by var-arg macros

Example

See the file `vaarg.c` in the *install_dir/docs/examples/* directory on your distribution.

va_start()

Initialize an object of type **va_list**

Targets

va_start() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdarg.h>    /* Required */
void va_start(va_list ap, parmN);
```

Description

va_start() initializes argument list *ap* using *parmN*, which must be the right-most named argument of the function whose unnamed arguments are desired, so that **va_arg()** can subsequently return the first unnamed argument.

va_start() must be called before the first call to **va_arg()**.

NOTE **va_start()** is provided as a macro only.

See Also

[va_arg\(\)](#) — Get the next argument in a variable series of arguments

[va_end\(\)](#) — Terminate va_list() processing

Type [va_list](#) — Information needed by var-arg macros

Example

See the file `vaarg.c` in the `install_dir/docs/examples/` directory on your distribution.

_vbprintf()

Print to a string of given size using var-arg macros

Targets

`_vbprintf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
#include <stdarg.h>
int _vbprintf(char *buf, unsigned int bufsize,
    const char *format, va_list arg ...);
```

Description

`_vbprintf()` formats data according to the format control string and writes the result to `buf`. The argument `bufsize` specifies the size of the character array `buf` into which the generated output is placed. See the section [Format String](#) in the description of function `printf()` for information about the format control string.

`_vbprintf()` is similar to `_bprintf()`, except that `_vbprintf()` takes a pointer to a list of arguments (`arg`), instead of an argument list, where `arg` has been initialized by the macro `va_start()` (and possibly updated by subsequent calls to `va_arg()`); for example:

```
_bprintf(buf, bufsize, format, ...);
_vbprintf(buf, bufsize, format, va_list arg);
```

See [stdarg.h](#) on page 39 for a discussion of var-arg (variable-argument) macros.

Return Value

`_vbprintf()` returns the number of characters written, or a negative value if an output error occurred.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[_bprintf\(\)](#) — Print to a string of given size

[printf\(\)](#) — Print to `stdout`

[va_arg\(\)](#) — Get the next argument in a variable series of arguments

[va_start\(\)](#) — Initialize an object of type va_list

Type [va_list](#) — Information needed by var-arg macros

_vdmemcpy()

Copy to a volatile destination (ARC only)

Targets

_vdmemcpy() works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <string.h>
extern void *_vdmemcpy(volatile void * __dest, const void * __source, size_t __n);
```

Description

_vdmemcpy() is an ARC-only version of **memcpy()** with a volatile destination. It copies n bytes from the memory location referenced by source to the volatile memory location referenced by dest, using the .di cache bypass for all stores. _vdmemcpy() copies from left to right (from low addresses to high addresses). Note that _vdmemcpy() copies the second argument to the first.

CAUTION _vdmemcpy() writes on n bytes.

Return Value

_vdmemcpy() returns a pointer to dest.

See Also

[memcpy\(\)](#) — Copy from one place in memory to another

[vmemset\(\)](#) — Duplicate a character to a volatile destination (ARC only)

[vsmemcpy\(\)](#) — Copy from a volatile source (ARC only)

[vsdmemcp\(\)](#) — Copy from a volatile source to a volatile destination (ARC only)

vfprintf()

Print to a file using var-arg macros

Targets

vfprintf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
int vfprintf(FILE *stream, const char *format, va_list arg);
```

Description

vfprintf() is similar to **fprintf()**, except that **vfprintf()** takes a pointer to a list of arguments (*arg*), instead of an argument list, where *arg* has been initialized by the macro **va_start()** (and possibly updated by subsequent calls to **va_arg()**). See [stdarg.h](#) on page 39 for a discussion of var-arg (variable-argument) macros.

CAUTION Garbage might be printed if the number of arguments passed to **vfprintf()** is less than the number specified by *format*, or if the types of the arguments to **vfprintf()** do not match the types specified by *format*.

Return Value

vfprintf() returns the number of characters written, or a negative value if an output error occurred.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[printf\(\) — Print to stdout](#)

[va_arg\(\) — Get the next argument in a variable series of arguments](#)

[va_start\(\) — Initialize an object of type va_list](#)

Type [va_list](#) — [Information needed by var-arg macros](#)

[typedef struct FILE — Information for controlling a stream](#)

Example

See the file `vfprintf.c` in the *install_dir/docs/examples/* directory on your distribution.

vfscanf() and _vfscanf()

Read values from a file using var-arg macros

Targets

`vfscanf()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_vfscanf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`vfscanf()` is Non-ANSI; `_vfscanf()` is Extra-ANSI.

Reentrancy

Both the functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>    /* Required */
#include <stdarg.h>
int vfscanf(const char *format, va_list arg);
int _vfscanf(const char *format, va_list arg);
```

Description

`vfscanf()` and `_vfscanf()` are similar to `fscanf()`, except that the `vfscanf()` and `_vfscanf()` functions take a pointer to a list of arguments (`arg`), instead of an argument list, where `arg` has been initialized by the macro `va_start()` (and possibly updated by subsequent `va_arg()` calls). See [stdarg.h](#) on page 39 for a discussion of var-arg (variable-argument) macros.

Return Value

`vfscanf()` and `_vfscanf()` return the number of input values assigned, or EOF if execution encounters end-of-file before the first conflict or conversion. If no conflict occurs, `vfscanf()` and `_vfscanf()` return when execution encounters the end of the format string.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[scanf\(\)](#) — [Read values from stdin](#)

[va_arg\(\)](#) — [Get the next argument in a variable series of arguments](#)

[va_start\(\)](#) — [Initialize an object of type va_list](#)

Type [va_list](#) — [Information needed by var-arg macros](#)

_vmemset()

Duplicate a character to a volatile destination (ARC only)

Targets

_vmemset() works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <string.h> /* Required */  
void *_vmemset(void volatile *start, int fill, size_t len);
```

Description

_vmemset() copies the character *fill* (the least significant byte of **int** *fill*) into each of *len* bytes starting at the volatile location pointed to by *start*.

CAUTION _vmemset() writes on *n* bytes.

Return Value

_vmemset() returns a pointer to *start*.

Surprises

_vmemset() copies the second argument to the first.

_vmemset() takes *fill* as an **int** rather than a **char** argument to conform to the conventions discussed in [Functions and Arguments](#) on page 23. If an object of type **char** is passed to _vmemset(), it is automatically coerced to type **int**.

See Also

[memset\(\)](#) — [Duplicate a character across an area of memory](#)

[vdmemcpy\(\)](#) — [Copy to a volatile destination \(ARC only\)](#)

[vsmemcpy\(\)](#) — [Copy from a volatile source \(ARC only\)](#)

[vsdmemcpy\(\)](#) — [Copy from a volatile source to a volatile destination \(ARC only\)](#)

Example

See the file `memset.c` in the *install_dir/docs/examples/* directory on your distribution.

vprintf()

Print to `stdout` using var-arg macros

Targets

vprintf() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Not-Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
int vprintf(const char *format, va_list arg);
```

Description

vprintf() is similar to **printf()**, except that **vprintf()** takes a pointer to a list of arguments (*arg*), instead of an argument list, where *arg* has been initialized by the macro **va_start()** (and possibly updated by subsequent calls to **va_arg()**). See [stdarg.h](#) on page 39 for a discussion of var-arg (variable-argument) macros.

CAUTION Garbage might be printed if the number of arguments passed to **vprintf()** is less than the number specified by *format*, or if the types of the arguments to **vprintf()** do not match the types specified by *format*.

Return Value

vprintf() returns the number of characters written, or a negative value if an output error occurred.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[printf\(\) — Print to stdout](#)

[va_arg\(\) — Get the next argument in a variable series of arguments](#)

[va_start\(\) — Initialize an object of type va_list](#)

Type [va_list — Information needed by var-arg macros](#)

_vsdmemcpy()

Copy from a volatile source to a volatile destination (ARC only)

Targets

_vsdmemcpy() works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <string.h>
extern void *_vsMemcpy(volatile void * __dest, const volatile void * __source, size_t __n);
```

Description

`_vsMemcpy()` is an ARC-only version of `memcpy()` with a volatile source and destination. It copies `__n` bytes from the volatile memory location referenced by `__source` to the volatile memory location referenced by `__dest`, using the `.di` cache bypass for all loads and stores. `_vsMemcpy()` copies from left to right (from low addresses to high addresses). Note that `_vsMemcpy()` copies the second argument to the first.

CAUTION `_vsMemcpy()` writes on `n` bytes.

Return Value

`_vsMemcpy()` returns a pointer to `__dest`.

See Also

[memcpy\(\)](#) — Copy from one place in memory to another
[vdMemcpy\(\)](#) — Copy to a volatile destination (ARC only)
[vmemset\(\)](#) — Duplicate a character to a volatile destination (ARC only)
[vsMemcpy\(\)](#) — Copy from a volatile source (ARC only)

_vsMemcpy()

Copy from a volatile source (ARC only)

Targets

`_vsMemcpy()` works on the following target platforms:

EMB	UNIX	DOS	NT
ARC	—	—	—

ANSI

Extra-ANSI

Reentrancy

Reentrant

Synopsis

```
#include <string.h>
extern void *_vsMemcpy(void * __dest, const volatile void * __source, size_t __n);
```

Description

`_vsMemcpy()` is an ARC-only version of `memcpy()` with a volatile source. It copies `__n` bytes from the volatile memory location referenced by `__source` to the memory location referenced by `__dest`, using the `.di` cache bypass for all loads. `_vsMemcpy()` copies from left to right (from low addresses to high addresses). Note that `_vsMemcpy()` copies the second argument to the first.

CAUTION `_vsMemcpy()` writes on n bytes.

Return Value

`_vsMemcpy()` returns a pointer to `__dest`.

See Also

[memcpy\(\)](#) — [Copy from one place in memory to another](#)

[_vdMemcpy\(\)](#) — [Copy to a volatile destination \(ARC only\)](#)

[_vmemset\(\)](#) — [Duplicate a character to a volatile destination \(ARC only\)](#)

[_vsdMemcpy\(\)](#) — [Copy from a volatile source to a volatile destination \(ARC only\)](#)

vscanf() and _vscanf()

Read values from `stdin` using var-arg macros

Targets

`vscanf()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_vscanf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`vscanf()` is Non-ANSI; `_vscanf()` is Extra-ANSI.

Reentrancy

Both of the functions are Not-Reentrant.

Synopsis

```
#include <stdio.h>           /* Required */
#include <stdarg.h>
int vscanf(const char *format, va_list arg);
int _vscanf(const char *format, va_list arg);
```

Description

`vscanf()` and `_vscanf()` are similar to `scanf()`, except that `vscanf()` and `_vscanf()` take a pointer to a list of arguments (`arg`), instead of an argument list, where `arg` has been initialized by the macro `va_start()` (and possibly updated by subsequent `va_arg()` calls). See [stdarg.h](#) on page 39 for a discussion of var-arg (variable-argument) macros.

Return Value

`vscanf()` and `_vscanf()` return the number of input values assigned, or EOF if execution encounters end-of-file before the first conflict or conversion. If no conflict occurs, `vscanf()` and `_vscanf()` return when execution encounters the end of the format string.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[scanf\(\)](#) — [Read values from stdin](#)

[va_arg\(\)](#) — [Get the next argument in a variable series of arguments](#)

[va_start\(\)](#) — [Initialize an object of type va_list](#)

Type [va_list](#) — [Information needed by var-arg macros](#)

vsprintf()

Print to a string using var-arg macros

Targets

`vsprintf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdio.h>           /* Required */
int vsprintf(char *s, const char *format, va_list arg);
```

Description

`vsprintf()` is similar to `sprintf()`, except that `vsprintf()` takes a pointer to a list of arguments (*arg*), instead of an argument list, where *arg* has been initialized by the macro `va_start()` (and possibly updated by subsequent calls to `va_arg()`). See [stdarg.h](#) on page 39 for a discussion of var-arg (variable-argument) macros.

CAUTION Garbage might be printed if the number of arguments passed to `vsprintf()` is less than the number specified by *format*, or if the types of the arguments to `vsprintf()` do not match the types specified by *format*.

Return Value

`vsprintf()` returns the number of characters written, or a negative value if an output error occurred.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[printf\(\)](#) — [Print to stdout](#)

[va_arg\(\)](#) — [Get the next argument in a variable series of arguments](#)

[va_start\(\)](#) — [Initialize an object of type va_list](#)

Type [va_list](#) — [Information needed by var-arg macros](#)

Example

See the file `vfprintf.c` in the `install_dir/docs/examples/` directory on your distribution.

vsscanf() and _vsscanf()

Read values from a string using var-arg macros

Targets

`vsscanf()` works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

`_vsscanf()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

`vsscanf()` is Non-ANSI; `_vsscanf()` is Extra-ANSI.

Reentrancy

Both the functions are Reentrant.

Synopsis

```
#include <stdio.h>           /* Required */
#include <stdarg.h>
int vsscanf(const char *s, const char *format, va_list arg);
int _vsscanf(const char *s, const char *format, va_list arg);
```

Description

`vsscanf()` and `_vsscanf()` are similar to `sscanf()`, except that `vsscanf()` and `_vsscanf()` take a pointer to a list of arguments (`arg`), instead of an argument list, where `arg` has been initialized by the macro `va_start()` (and possibly updated by subsequent `va_arg()` calls). See [stdarg.h](#) on page 39 for a discussion of var-arg (variable-argument) macros.

Return Value

`vsscanf()` and `_vsscanf()` return the number of input values assigned, or EOF if execution encounters end-of-file before the first conflict or conversion. If no conflict occurs, `vsscanf()` and `_vsscanf()` return when execution encounters the end of the format string.

See Also

[The *printf Family of Functions](#) on page 424

[The *scanf Family of Functions](#) on page 424

[scanf\(\) — Read values from stdin](#)

[va_arg\(\) — Get the next argument in a variable series of arguments](#)

[va_start\(\) — Initialize an object of type va_list](#)

Type [va_list — Information needed by var-arg macros](#)

wcstombs()

Convert a `wchar_t` array to a multibyte string

Targets

`wcstombs()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
```

Description

`wcstombs()` converts the array of values of type `wchar_t` pointed to by `pwcs` into the multibyte string `s`, until a multibyte character would exceed the limit of `n` total bytes or until a null character is stored.

NOTE If `wcstombs()` stores `n` bytes before encountering a null character, `s` is not null-terminated.

Return Value

If successful, `wcstombs()` returns the number of bytes modified. No more than `n` bytes are modified. If an invalid multibyte character is encountered during conversion, `wcstombs()` returns `(size_t)-1`.

See Also

[mblen\(\)](#) — [Determine length of a multibyte character](#)

[mbstowcs\(\)](#) — [Convert a multibyte string to wchar_t array](#)

[mbtowc\(\)](#) — [Convert a multibyte character to wchar_t code](#)

[wctomb\(\)](#) — [Convert a wchar_t code to a multibyte character](#)

Type [wchar_t](#) — [Integral type; extended-character-set values](#)

Constant [MB_CUR_MAX](#) — [Maximum number of bytes for a character in current locale](#)

Constant [MB_LEN_MAX](#) — [Maximum number of bytes in a multibyte character](#)

wctomb()

Convert a `wchar_t` code to a multibyte character

Targets

`wctomb()` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
#include <stdlib.h>           /* Required */
int wctomb(char *s, wchar_t wchar);
```

Description

If *s* is not **NULL**, **wctomb()** determines the number of bytes needed to represent the multibyte character corresponding to *wchar* (including any change in shift state) and stores the multibyte character in *s*. At most **MB_CUR_MAX** bytes are stored. If the value of *wchar* is 0 (zero), **wctomb()** is left in the initial shift state.

Return Value

If *wchar* is a valid code, **wctomb()** returns the number of bytes that comprise the multibyte character. If *wchar* is an invalid code, **wctomb()** returns -1.

If *s* is **NULL** and multibyte characters have state-dependent encodings, **wctomb()** returns a non-zero value. If *s* is **NULL** and multibyte characters do not have state-dependent encodings, **wctomb()** returns 0 (zero).

See Also

[mblen\(\)](#) — [Determine length of a multibyte character](#)

[mbstowcs\(\)](#) — [Convert a multibyte string to wchar_t array](#)

[mbtowc\(\)](#) — [Convert a multibyte character to wchar_t code](#)

[westombs\(\)](#) — [Convert a wchar_t array to a multibyte string](#)

Type [wchar_t](#) — [Integral type; extended-character-set values](#)

Constant [MB_CUR_MAX](#) — [Maximum number of bytes for a character in current locale](#)

Constant [MB_LEN_MAX](#) — [Maximum number of bytes in a multibyte character](#)

write() and _write()

Write data to a file, unbuffered

Targets

write() works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

_write() works on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

write() is Non-ANSI; **_write()** is Extra-ANSI.

Reentrancy

Both the functions are Not-Reentrant

Synopsis

```
/* Required for DOS and Windows targets: */
#include <io.h>
/* Required for all other targets: */
#include <unistd.h>

int write(int handle, char *bufptr, unsigned int count);
int _write(int handle, char *bufptr, unsigned int count);
```

Description

write() and **_write()** write *count* bytes, starting at *bufptr*, into the file currently opened with *handle*. The write occurs at the current position in the file, unless the file is open for append, in which case the data is written at the current end of the file. After the write, the file is positioned one byte after the last byte written.

Return Value

On success, **write()** and **_write()** return the actual number of bytes taken from the buffer (at *bufptr*). The return value might be less than *count*, if the write operation could not be completed (disk was full).

If an error occurs, **write()** and **_write()** return -1 and set global variable **errno** to one of the following values:

[EBADF — “Bad” \(invalid or inaccessible\) file handle](#)

[ENOSPC — No space left on device](#) to start the write operation

The return value does not include the carriage-return characters added when the file is in text mode.

System issues

When writing to some character devices, DOS treats the CTRL-Z character as an end-of-file code, and stops writing at that point.

On UNIX, this function can be provided by the system library.

See Also

[I/O Functions](#) on page 427 ([File I/O](#))

[File-Related Functions](#) on page 429

Global variable [errno](#) — [An int used to record error numbers](#)

Example

See the file `write.c` in the *install_dir/docs/examples/* directory on your distribution.

y0(), y1(), yn() and _y0(), _y1(), _yn()

Bessel function of the second kind

Targets

The **y***() functions work on the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

The `_y*`() functions work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The `y*`() functions are Non-ANSI; the `_y*`() functions are Extra-ANSI.

Reentrancy

All these functions are Reentrant.

Synopsis

```
#include <math.h>

double y0(double x);
double y1(double x);
double yn(int pow, double x);

double _y0(double x);
double _y1(double x);
double _yn(int pow, double x);
```

Description

`y0()` and `_y0()` calculate a Bessel function of the second kind, power 0.

`y1()` and `_y1()` calculate a Bessel function of the second kind, power 1.

`yn()` and `_yn()` calculate a Bessel function of the second kind, power integral `pow`.

Return Value

The result of `y0()` and `_y0()` is a Bessel function of `x` to power 0.

The result of `y1()` and `_y1()` is a Bessel function of `x` to power 1.

The result of `yn()` and `_yn()` is a Bessel function of `x` to power integral `pow`.

In any of the `y*`() or `_y*`() functions, if $x < 0$, the function returns [HUGE_VAL](#) and sets `errno` to [EDOM](#) ([Mathematical domain error](#)).

See Also

[`j0\(\)`, `j1\(\)`, `jn\(\)` and `_j0\(\)`, `_j1\(\)`, `_jn\(\)` — Bessel function of the first kind](#)

Example

See the file `j.c` in the `install_dir/docs/examples/` directory on your distribution.

Chapter 5 — Constants and Globals

In This Chapter

- [*Listing of Constants and Global Variables*](#)

Listing of Constants and Global Variables

This section lists the constants and global variables in alphabetical order.

A_*

File attributes

Targets

The A_* constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The A_* constants are Extra-ANSI.

Synopsis

```
#include <io.h> /* Required */
```

Description

The A_* constants represent the attributes of a file.

A_ARCH

Archive file; this attribute is set whenever the file is changed. It is cleared by the DOS **backup** command.

A_HIDDEN

Hidden file; a file with this attribute will not be found by a directory search.

A_NORMAL

Normal file; the file can be read from or written to.

A_RDONLY

Read only file; the file cannot be opened in write mode, and a file with the same name cannot be created.

A_SYSTEM

System file; a file with this attribute is also invisible to a directory search.

A_SUBDIR

Subdirectory; the specified pathname refers to a subdirectory.

A_VOLID

Volume ID; only one file can have this attribute, and it must be located in the root directory.

See Also

[fclose\(\) and _findfirst\(\) and _findnext\(\)](#) — Find a file whose name matches a specification

[struct _finddata_t](#) — Attributes of a file

BUFSIZ

Size of a buffer used by **setbuf()**

Targets

Constant **BUFSIZ** is defined on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h> /* Required */
```

Description

BUFSIZE is an integer constant representing the size (in bytes) of the buffers used by the **setbuf()** function.

See Also

[setbuf\(\)](#) — [Specify a buffer for a stream](#)

[File-Related Functions](#) on page 429

CHAR_*

Minimum and maximum values for **char** types

Targets

The **CHAR_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **CHAR_*** constants are ANSI compliant.

Synopsis

```
#include <limits.h> /* Required */
```

Description

The **CHAR_*** constants provide minimum and maximum values for **char** types.

CHAR_BIT

Maximum number of bits for smallest object (byte)

CHAR_MAX

Maximum value for an object of type **char**

CHAR_MIN

Minimum value for an object of type **char**

CLOCKS_PER_SEC

Clock ticks converted to seconds

Targets

Constant CLOCKS_PER_SEC is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <time.h> /* Required */
```

Description

CLOCKS_PER_SEC is a macro constant used to convert the time units provided by the operating system to seconds.

See Also

[clock\(\)](#) — Report elapsed processor time

[clock_t](#) — Representation of elapsed processor time

[time\(\)](#) — Get the current time and date

COM*

Constants for BIOS asynchronous communication

Targets

The _COM_* constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	—

ANSI

The _COM_* constants are Extra-ANSI.

Synopsis

```
#include <bios.h> /* Required */
```

Description

The _COM_* constants are used in calls to [_bios_serialcom\(\)](#) to specify asynchronous communication parameters.

_COM_INIT

Sends an argument byte to the specified serial port

_COM_RECEIVE

Accepts an input character from the specified serial port

_COM_SEND

Transmits the data character over the specified serial port

COM_STATUS

Returns the current status of the specified serial port

With service **COM_STATUS**, the argument byte is constructed by bitwise ORing one or more of the following constants with the byte used as the argument data.

- Data-bit constants (use one):
COM_CHR7 (7 data bits) or
COM_CHR8 (8 data bits)
- Stop-bit constants (use one):
COM_STOP1 (1 stop bit) or
COM_STOP2 (2 stop bits)
- Parity constants (use one):
COM_NOPARITY (No parity)
COM_EVEN_PARITY (Even parity)
COM_ODDPARITY (Odd parity)
- Baud-rate constants (use one):
COM_110 (110 baud),
COM_150 (150 baud),
COM_300 (300 baud),
COM_600 (600 baud),
COM_1200 (1,200 baud),
COM_2400 (2,400 baud),
COM_4800 (4,800 baud),
COM_9600 (9,600 baud),
COM_19_2 (19,200 baud),
COM_38_4 (38,400 baud),
COM_57_6 (57,600 baud),
COM_115_2 (115,200 baud)

NOTE Reading or writing strings as separate characters at speeds greater than 300 baud is not recommended. For interactive communications at higher speeds, consider using an interrupt-driven system from a specialized vendor.

DATE

File date of translation

Targets

Predefined macro **DATE** is supported on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

_daylight

Reentrancy

Not-Reentrant

Synopsis

```
__DATE__ /* Predefined macro */
```

Description

`__DATE__` produces the date of translation of the source file as a string of this form:

mmm dd yyyy

- *mmm* is the month, spelled as Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec.
- *dd* is the day (0 to 31). If the value of *dd* is less than 10, the first character of *dd* is a space character.
- *yyyy* is the year (0000 to 9999).

See Also

[TIME](#) — [File time of translation](#)

[Time-Related Functions](#) on page 434

_daylight

Pacific Daylight Time

Targets

Variable `_daylight` is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Variable `_daylight` is Extra-ANSI.

Synopsis

```
#include <time.h> /* Required */
#include <static.h> /* Required */
```

Description

Function `tzset()` or `_tzset()` sets global variable `_daylight` to 1 (one) if PDT is present, and to 0 (zero) if PDT is not present. (PDT is a possible value for the DDD component of environment variable TZ.)

See Also

[Time-Related Functions](#) on page 434

[tzset\(\) and _tzset\(\)](#) — [Set local time-zone values](#)

[Environment Variable TZ](#)

DBL_*

Minimum and maximum values for **double** calculations

Targets

The **DBL_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **DBL_*** constants are ANSI compliant.

Synopsis

```
#include <float.h> /* Required */
```

Description

The **DBL_*** constants provide minimum and maximum values for calculations involving **double** numbers.

DBL_DIG

Maximum number of decimal digits of **float** precision in a **double** number

DBL_EPSILON

Minimum positive **double** number x such that $1.0 + x \neq 1.0$

DBL_MANT_DIG

Number of base **DBL_RADIX** digits in mantissa

DBL_MAX

Maximum representable finite **double** number

DBL_MAX_10_EXP

Maximum integer such that 10 raised to that power is in the range of representable finite **double** numbers

DBL_MAX_EXP

Maximum integer such that **FLT_RADIX** raised to that power minus 1 is a representable finite **double** number

DBL_MIN

Minimum normalized positive **double** number

DBL_MIN_10_EXP

Minimum negative integer such that 10 raised to that power is in the range of normalized **double** numbers

DBL_MIN_EXP

Minimum negative integer such that **FLT_RADIX** raised to that power minus 1 is a normalized **double** number

DBL_RADIX

Radix for **double** mantissa representation

See Also

[FLT_*](#) — Minimum and maximum values for float calculations

[**LDBL**](#) * — Minimum and maximum values for long double calculations

_E

Natural logarithm base e

Targets

Macro constant **_E** is supported on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

Extra-ANSI

Synopsis

```
#include <math.h> /* Required */  
#define _E 2.718281828459045235360287471352662497757247093
```

Description

This macro produces the value of the natural logarithm with base e.

EDOM

Mathematical domain error

Targets

Constant **EDOM** is supported on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Synopsis

```
#include <errno.h> /* Required */
```

Description

EDOM expands to an integer constant that is the error code for a domain error. Several functions set global variable **errno** to **EDOM** when an argument is not in the mathematical domain over which the function is defined. **EDOM** is defined in **math.h** and **errno.h**.

EOF

End-of-file indicator

Targets

Constant **EOF** is supported on the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h>          /* Required */
```

Description

EOF expands to a negative integer used to indicate end-of-file.

ERANGE

Mathematical range error

Targets

Constant **ERANGE** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <errno.h>    /* Required */
```

Description

ERANGE expands to an integer constant that is the error code for a range error. Several functions set global variable **errno** to **ERANGE** when a result is not in the range of the return type of the function (that is, the result is too large in magnitude to fit into the function's return type). **ERANGE** is defined in **math.h** and **errno.h**.

errno

An **int** used to record error numbers

Targets

Global variable **errno** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <errno.h>    /* Required */
```

Description

Several functions reference the **int** variable **errno**. To reference **errno** from a program, you must **#include** the header file **errno.h**, or declare **errno** to be external (**extern int errno;**).

Some library functions set **errno** to a positive integer error code when an error occurs during their execution. **errno** is set to 0 (zero) on program start-up; it is never set to 0 (zero) by any library function.

Many functions write to the static area used by `errno`. If the program ignores `errno`, the overwriting does not interfere with reentrancy.

NOTE Other error codes might be defined in a system-dependent manner, and not all of these error codes might actually be generated by any particular operating system. For example, not all of the error codes listed here are implemented on DOS.

Error Codes for Global Variable `errno`

The following list summarizes the error codes defined for `errno`. All these values are ANSI-compliant.

E2BIG	Argument list too long
EACCES	Access error; permission denied The requested mode of access is not allowed by the file's permission setting.
EAGAIN	No more processes or too many threads
EBADF	"Bad" (invalid or inaccessible) file handle
EBUSY	Mount device busy
ECHILD	Invalid process specified, or process has no children
EDEADLOCK	Network cannot be accessed
EDOM	Domain error; argument too large
EEXIST	File already exists, so it cannot be created
EFAULT	Bad address
EFBIG	File too large
EINTR	Interrupted system call
EINVAL	Invalid argument given
EIO	I/O error
EISDIR	Argument is a directory
EMFILE	Too many open file handles
EMLINK	Too many links
ENFILE	File-table overflow
ENODEV	No such device
ENOENT	File or directory not found; pathname element does not exist
ENOEXEC	EXEC format error
ENOMEM	Not enough memory, or invalid DOS arena headers
ENOSPC	No space left on device
ENOTBLK	Block device required
ENOTDIR	Not a directory
ENOTTY	Not a teletype
ENXIO	No such device or address
EPERM	Not owner
EPIPE	Broken pipe
ERANGE	Result too large
EROFS	Read-only file system
ESPIPE	Invalid seek
ESRCH	No such process
ETXTBSY	Text file busy

EXDEV	Cross-device link
EZERO	No error

EXIT_FAILURE and EXIT_SUCCESS

Arguments to `exit()`

Targets

The `EXIT_*` macro constants work on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdlib.h> /* Required */
```

Description

Each `EXIT_*` macro expands to an integer to be used as the argument to `exit()` to indicate either successful or unsuccessful program termination:

EXIT_FAILURE

Program is terminating unsuccessfully.

EXIT_SUCCESS

Program is terminating successfully.

See Also

[abort\(\)](#) — Terminate a program abnormally

[exit\(\)](#) — Terminate a program normally

__FILE__

Name of the current source file

Targets

Predefined macro `__FILE__` is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
__FILE__ /* Predefined macro */
```

Description

`__FILE__` is useful in code debugging.

See Also

[LINE](#) — Line number of the current source line

Example

This is a typical debugging line:

```
printf("Now at __FILE__ __LINE__ "\n");
```

FILENAME_MAX

Maximum number of characters in a file name

Targets

Constant FILENAME_MAX is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h> /* Required */
```

Description

Expands to an integer representing the maximum number of characters in a valid file name.

See Also

[File-Related Functions](#) on page 429

FLT_*

Minimum and maximum values for **float** calculations

Targets

The FLT_* constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The FLT_* constants are ANSI compliant.

Synopsis

```
#include <float.h> /* Required */
```

Description

The FLT_* constants provide minimum and maximum values for calculations involving **float** numbers.

FLT_DIG

Maximum number of decimal digits of precision in a **float** number

FLT_EPSILON

Minimum positive **float** number x such that:

$$1.0 + x \neq 1.0$$

FLT_MANT_DIG

Number of base **FLT_RADIX** digits in mantissa

FLT_MAX

Maximum representable finite **float** number

FLT_MAX_10_EXP

Maximum integer such that 10 raised to that power is in the range of representable finite **float** numbers

FLT_MAX_EXP

Maximum integer such that **FLT_RADIX** raised to that power minus 1 is a representable finite **float** number

FLT_MIN

Minimum normalized positive **float** number

FLT_MIN_10_EXP

Minimum negative integer such that 10 raised to that power is in the range of normalized **float** numbers

FLT_MIN_EXP

Minimum negative integer such that **FLT_RADIX** raised to that power minus 1 is a normalized **float** number

FLT_RADIX

Radix of **float** exponent representation

FLT_ROUNDS

Rounding mode of **float** addition; 1 (one) = rounds, 0 (zero) = truncates

See Also

[**DBL_*** — Minimum and maximum values for double calculations](#)

[**LDBL_*** — Minimum and maximum values for long double calculations](#)

FOPEN_MAX

Maximum number of open files

Targets

Constant **FOPEN_MAX** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h>           /* Required */
```

Description

FOPEN_MAX expands to the maximum number of files that can be open at one time, including the five standard files. This value is independent of any system limit on the maximum number of open files.

NOTE The library is designed to be independent of system-imposed limits on the maximum number of open files. However, this does not preclude those limits from being reached.

System issues

The maximum number of open files in DOS is 50. This limit can be raised by using DOS system call **int 21h** with function **67H**.

On DOS and Windows NT, the limit can be adjusted by modifying the value of **FOPEN_MAX** in the **highc\lib\src_\io.b.c** file, compiling the file, and linking with its object file.

On UNIX targets, you might want to consider the file-open limits defined in the UNIX-supplied header files. Macro names include **NFILE** and **NOFILE**.

_HEAP*

State of the heap

Targets

The **_HEAP*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **_HEAP*** constants are Extra-ANSI.

Synopsis

```
#include <malloc.h>    /* Required */
```

Description

The **_HEAP*** constants are integer values that represent the state of the heap. These constant values are returned by the functions **_heapchk()**, **_heapset()**, and **_heapwalk()**.

_HEAPBADBEGIN

Initial header information not found or incorrect.

_HEAPBADNODE

Heap is damaged or a bad node was found.

_HEAPEMPTY

Heap was not initialized.

_HEAPEND

Reached the end of heap. (Returned only by **_heapwalk()**.)

_HEAPOK

Heap is consistent.

See Also

[heapchk\(\)](#) — Check heap for consistency

[heapset\(\)](#) — Fill free memory locations in the heap

[heapwalk\(\)](#) — Walk through the heap

--HIGHC--

Macro defined for implementations of MetaWare C not compiling in ANSI mode

Targets

Predefined macro **--HIGHC--** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

`--HIGHC-- /* Predefined macro */`

Description

The predefined macro **--HIGHC--** can effectively exclude code that depends upon features of MetaWare C that are not ANSI-supported.

--HIGHC-- is predefined to be 1 (one) for all implementations of MetaWare C unless you are compiling in ANSI mode.

HUGE_VAL

Largest positive floating-point value

Targets

Constant **HUGE_VAL** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

`#include <math.h> /* Required */`

Description

HUGE_VAL expands to a **double** constant that is the largest positive floating-point value that fits in a **double**.

INT_*

Several functions return **HUGE_VAL** when a valid result cannot be produced.

Example

See the file `hugeval.c` in the `install_dir/docs/examples/` directory on your distribution.

INT_*

Minimum and maximum values for **int** types

Targets

The **INT_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

The **INT_*** constants are ANSI compliant.

Synopsis

```
#include <limits.h> /* Required */
```

Description

The **INT_*** constants define minimum and maximum values for **int** types.

INT_MAX

Maximum value for an object of type **int**

INT_MIN

Minimum value for an object of type **int**

_INTERRPT

Designation of a DOS interrupt handler

Targets

Constant **_INTERRPT** is supported on the following target platforms:

EMB	UNIX	DOS	NT
—	—	X	—

ANSI

Extra-ANSI

Synopsis

```
#include <dos.h> /* Required */
```

Description

_INTERRPT is a macro used for setting up the calling convention for an interrupt handler.

_IOFBF and _IOLBF and _IONBF

Buffering-mode arguments to **setvbuf()**

Targets

The **_IO*BF** macros are supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **_IO*BF** macros are ANSI compliant.

Synopsis

```
#include <stdio.h> /* Required */
```

Description

Each **_IO*BF** macro expands to an integer to be used as the third argument to **setvbuf()**, indicating the buffering mode. The macros are defined as follows:

_IOFBF

Full buffering

_IOLBF

Line buffering

_IONBF

No buffering

See Also

[setvbuf\(\)](#) — Control file buffering

[File-Related Functions](#) on page 429

KEYBRD*

Keyboard services for function **_bios_keybrd()**

Targets

The **_KEYBRD_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	—

ANSI

The **_KEYBRD_*** constants are Extra-ANSI.

Synopsis

```
#include <bios.h> /* Required */
```

Description

The **_KEYBRD_*** constants define BIOS keyboard services for use in calls to function **_bios_keybrd()**.

_KEYBRD_READ

Reads the next character from the keyboard input buffer.

LC_*

If no character has been typed, the call to `_bios_keybrd()` waits for one. If the low-order byte of the return value is not zero, it contains the ASCII value of the character typed. The high-order byte contains the keyboard scan code for that character. See the **Technical Reference Manual** for your machine for a table of keyboard scan codes.

_KEYBRD_READY

Checks for a keystroke waiting in the keyboard buffer.

If a keystroke is waiting, `_bios_keybrd()` reads the keystroke but leaves it in the buffer. If no keystroke is waiting, the return value is 0 (zero); otherwise, the return value is the value of the keystroke found.

_KEYBRD_SHIFT_STATUS

Returns the current special key status in the low-order byte of the return value.

If the return value is 0 (zero), no special keys have been held or toggled. If the return value is non-zero, the non-zero bits have the following meanings:

- Bit 0: Mask = 1 (Right-SHIFT pressed)
- Bit 1: Mask = 2 (Left-SHIFT pressed)
- Bit 2: Mask = 4 (CTRL key pressed)
- Bit 3: Mask = 8 (ALT key pressed)
- Bit 4: Mask = 16 (Scroll Lock is on)
- Bit 5: Mask = 32 (Num Lock is on)
- Bit 6: Mask = 64 (Caps Lock is on)
- Bit 7: Mask = 128 (Insert mode is on)

LC_*

Arguments to `setlocale()`

Targets

The **LC_*** macros are supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **LC_*** macros are ANSI compliant.

Synopsis

```
#include <locale.h>           /* Required */
```

Description

Each **LC_*** macro expands to an integer to be used as the first argument to the `setlocale()` function. These arguments designate which parts of the program's locale are to be changed:

LC_ALL

Affects the entire locale.

LC_COLLATE

Affects the **strcoll()** and **strxfrm()** functions.

LC_CTYPE

Affects character and multibyte functions.

LC_MONETARY

Affects monetary numeric formatting.

LC_NUMERIC

Affects non-monetary numeric formatting.

LC_TIME

Affects the **strftime()** function.

See Also

[setlocale\(\)](#) — Set the current locale

[struct lconv](#) — Locale-specific numeric representation

LDBL_*

Minimum and maximum values for **long double** calculations

Targets

The **LDBL_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

The **LDBL_*** constants are ANSI compliant.

Synopsis

```
#include <float.h> /* Required */
```

Description

The **LDBL_*** constants provide minimum and maximum values for calculations involving **long double** numbers.

LDBL_DIG

Maximum number of decimal digits of precision in a **long double** number

LDBL_EPSILON

Minimum positive **long double** number *x* such that:

```
1.0 + x != 1.0
```

LDBL_MANT_DIG

Number of base **LDBL_RADIX** digits in mantissa

LDBL_MAX

Maximum representable finite **long double** number

LDBL_MAX_10_EXP

__LINE__

Maximum integer such that 10 raised to that power is in the range of representable finite **long double** numbers

LDBL_MAX_EXP

Maximum integer such that **FLT_RADIX** raised to that power minus 1 is a representable finite **long double** number

LDBL_MIN

Minimum normalized positive **long double** number

LDBL_MIN_10_EXP

Minimum negative integer such that 10 raised to that power is in the range of normalized **long double** numbers

LDBL_MIN_EXP

Minimum negative integer such that **FLT_RADIX** raised to that power minus 1 is a normalized **long double** number

LDBL_RADIX

Radix for **long double** mantissa representation

See Also

[DBL_*](#) — Minimum and maximum values for double calculations

[FLT_*](#) — Minimum and maximum values for float calculations

__LINE__

Line number of the current source line

Targets

Predefined macro __LINE__ is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

`__LINE__ /* Predefined macro */`

Description

__LINE__ expands to a decimal digit string whose value is the line number of the source line containing the occurrence of __LINE__. This macro is useful in code debugging.

See Also

[FILE](#) — Name of the current source file

Example

This is a typical debugging line:

```
printf("Now at __FILE__ __LINE__ '\n');
```

LK_* and _LK_*

File-locking mode

Targets

The LK_* and _LK_* constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The LK_* constants are Non-ANSI; the _LK_* constants are Extra-ANSI.

Synopsis

```
#include <locking.h> /* Required */
```

Description

The LK_* and _LK_* constants control the locking action of functions **locking()** and **_locking()**.

LK_LOCK

_LK_LOCK

Locks the indicated bytes. If the lock request fails, it is retried after a short pause (about one second). After ten locking failures, the function returns with an error indication.

LK_NBLCK

_LK_NBLCK

Locks the indicated bytes. If the lock request fails, an error is indicated in the calling function's return value.

LK_NBRLCK

_LK_NBRLCK

Performs the same function as **_LK_NBLCK**.

LK_RLCK

_LK_RLCK

Performs the same function as **_LK_LOCK**. Included for compatibility.

LK_UNLCK

_LK_UNLCK

Unlocks the indicated bytes. (A successful identical locking request must have been previously made.)

LONG_*

Minimum and maximum values for **signed long** types

Targets

The **LONG_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **LONG_*** constants are ANSI compliant.

Synopsis

```
#include <limits.h> /* Required */
```

Description

The **LONG_*** constants provide minimum and maximum values for **signed long** types.

LONG_MAX

Maximum value for an object of type **signed long**

LONG_MIN

Minimum value for an object of type **signed long**

L_tmpnam

Size of a temporary file name

Targets

Constant **L_tmpnam** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h> /* Required */
```

Description

L_tmpnam is an integer constant expression of the size (in bytes) of an array large enough to hold a temporary file name generated by a call to **tmpnam()**.

See Also

[tmpnam\(\) — Generate a string to be used as a temporary file name](#)

[File-Related Functions on page 429](#)

MAX_*

Maximum length of several pathname components

Targets

The _MAX_* constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The _MAX_* constants are Extra-ANSI.

Synopsis

```
#include <stdlib.h> /* Required */
```

Description

The _MAX_* constants specify the maximum sizes of the buffers used by the functions _makepath() and _splitpath() for pathname components.

_MAX_DIR

Maximum length of directory part of path.

_MAX_DRIVE

Maximum length of drive identifier.

_MAX_EXT

Maximum length of file extension.

_MAX_FNAME

Maximum length of file name.

_MAX_PATH

Maximum length of entire pathname.

See Also

[_makepath\(\)](#) — Create a path string from components

[_splitpath\(\)](#) — Split a pathname into its components

_MAXSTRING

Maximum length of a string

Targets

Constant _MAXSTRING is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Synopsis

```
#include <string.h> /* Required */
```

Description

_MAXSTRING expands to the maximum possible length of a string.

MB_CUR_MAX

Maximum number of bytes for a character in current locale

Targets

Constant **MB_CUR_MAX** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdlib.h> /* Required */  
#define MB_CUR_MAX 1 /* Default value in stdlib.h */
```

Description

MB_CUR_MAX expands to the maximum number of bytes for a character in the current locale.

See Also

[mblen\(\)](#) — Determine length of a multibyte character

[mbstowcs\(\)](#) — Convert a multibyte string to wchar_t array

[mbtowc\(\)](#) — Convert a multibyte character to wchar_t code

[wcstombs\(\)](#) — Convert a wchar_t array to a multibyte string

[wctomb\(\)](#) — Convert a wchar_t code to a multibyte character

Type [wchar_t](#) — Integral type; extended-character-set values

Constant [MB_LEN_MAX](#) — Maximum number of bytes in a multibyte character

MB_LEN_MAX

Maximum number of bytes in a multibyte character

Targets

Constant **MB_LEN_MAX** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant.

Synopsis

```
#include <limits.h> /* Required */
```

Description

MB_LEN_MAX is defined as the maximum number of bytes in a multibyte character.

See Also

[mblen\(\)](#) — Determine length of a multibyte character

[mbstowcs\(\)](#) — Convert a multibyte string to wchar_t array

[**mbtowc\(\)**](#) — Convert a multibyte character to wchar_t code

[**westombs\(\)**](#) — Convert a wchar_t array to a multibyte string

[**wctomb\(\)**](#) — Convert a wchar_t code to a multibyte character

Type [**wchar_t**](#) — Integral type; extended-character-set values

Constant [**MB_CUR_MAX**](#) — Maximum number of bytes for a character in current locale

NULL

The null pointer

Targets

Constant **NULL** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h>           /* Required */
```

Description

NULL expands to a value that is assignment-compatible with any data pointer type and compares equal with the constant **0** (zero). It is therefore suitable as a representation of the null pointer.

CAUTION Note that **NULL** is not appropriate as the terminating character of a string, because the size of a pointer is not the same as the size of the character **NUL**.

NULL is defined in `locale.h`, `stddef.h`, `stdio.h`, `stdlib.h`, `string.h`, and `time.h`.

O_* and _O_*

File-open mode flags

Targets

The **O_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

The **_O_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The **O_*** constants are Non-ANSI; the **_O_*** constants are Extra-ANSI.

Synopsis

```
#include <fcntl.h> /* Required */
```

Description

The **O_*** and **_O_*** constants are flags that specify the “open mode” for files opened by the functions **_dos_open()**, **open()**, and **sopen()**. Compatible flags can be joined together with the OR operator (**|**).

O_APPEND

_O_APPEND

Opens the file for appending. This causes all writes to be performed at the end of the file, thus extending what is already in the file.

O_BINARY

_O_BINARY

Opens the file for binary data. No special character translation is performed. All characters are passed through unchanged.

O_CREAT

_O_CREAT

Causes the file to be created if it does not already exist. This flag is ignored if the file already exists.

O_EXCL

_O_EXCL

Causes an error return if the file already exists. The file is opened only if it is a new file. Can be specified only with **_O_CREAT**.

O_NOINHERIT

_O_NOINHERIT

Specifies that the file is not inherited by any child processes; a spawned task should not inherit this file.

O_RDONLY

_O_RDONLY

Specifies that the file should not be written to; only reading will be performed.

O_RDWR

_O_RDWR

Specifies that the file can be written to or read from.

O_TEXT

_O_TEXT

Opens the file with text translation enabled. This converts the C line terminator **\n** to (or from) the DOS line terminator **\r\n**. It also enables the CTRL-Z character as an end-of-file marker.

O_TRUNC

_O_TRUNC

Destroys any existing file before opening the pathname specified. Because this in effect writes to the file, the file must allow writing.

O_WRONLY

_O_WRONLY

Specifies that the file will not be read from, but will only be written to.

See Also

[open\(\) and _open\(\) — Open a file handle with a pathname](#)

P^{*}

Determine mode of child and behavior of parent

Targets

The P^{*} constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The P^{*} constants are Extra-ANSI.

Synopsis

```
#include <process.h> /* Required */
```

Description

The P^{*} constants, used by the **spawn*()** and **_spawn*()** functions, determine how the child program is loaded and how the calling (parent) program behaves.

_P_WAIT

The parent process is suspended until completion of the child process.

_P_NOWAIT

The parent process continues execution concurrently with the child process.

_P_NOWAITO

The parent process continues execution concurrently with the child process. The child process ID is not saved, so the parent process cannot wait for it using **_wait()**.

_P_DETACH

The same as **_P_NOWAITO**, except the child process is executed in the background. The child process has no access to the keyboard or the display.

_P_OVERLAY

The child process replaces the parent process.

See Also

[spawn*\(\) and _spawn*\(\) — Execute a child process](#)

_PI

Value of pi

Targets

Macro constant _PI is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Synopsis

```
#include <math.h>          /* Required */  
#define _PI  
    3.141592653589793238462643383279502884197169399
```

Description

The _PI macro produces the value for pi.

PRINTER*

BIOS printer services

Targets

The _PRINTER_* constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	—

ANSI

The _PRINTER_* constants are Extra-ANSI.

Synopsis

```
#include <bios.h>    /* Required */
```

Description

The _PRINTER_* constants define available direct printer services for function _bios_printer().

_PRINTER_INIT

Initializes the selected printer.

All states are set to default. *data* is ignored, and the printer status is returned.

_PRINTER_STATUS

Checks status only.

data is ignored, and the printer status is returned.

_PRINTER_WRITE

Sends the low-order byte of data to the printer specified by *port*.

With the **_PRINTER_WRITE** service, the low-order byte of the return value contains the bit pattern for the status of the printer after the operation. When a bit is set to 1 (one), it has one of the following meanings:

- Bit 0: Mask = 1 (0000 0001) — Printer timed out
- Bit 1: Mask = 2 (0000 0010) — Not used
- Bit 2: Mask = 4 (0000 0100) — Not used
- Bit 3: Mask = 8 (0000 1000) — I/O error
- Bit 4: Mask = 16 (0001 0000) — Printer selected
- Bit 5: Mask = 32 (0010 0000) — Out of paper
- Bit 6: Mask = 64 (0100 0000) — Acknowledge (OK)
- Bit 7: Mask = 128 (1000 0000) — Printer not busy

RAND_MAX

Maximum random number

Targets

Constant **RAND_MAX** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdlib.h>
#define RAND_MAX 32767
```

Description

RAND_MAX specifies the maximum random number.

SCHAR_*

Minimum and maximum values for **signed char** types

Targets

The **SCHAR_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **SCHAR_*** constants are ANSI compliant.

Synopsis

```
#include <limits.h> /* Required */
```

Description

The SCHAR_* constants provide minimum and maximum values for **signed char** types.

SCHAR_MAX

Maximum value for an object of type **signed char**

SCHAR_MIN

Minimum value for an object of type **signed char**

SEEK_CUR and SEEK_END and SEEK_SET

Macros used to specify position in file

Targets

The SEEK_* macros are supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The SEEK_* constants are ANSI compliant.

Synopsis

```
#include <stdio.h> /* Required */
```

Description

Each SEEK_* macro expands to an integer to be used as the third argument to **fseek()** or **Iseek()**, indicating where **fseek()** or **Iseek()** should seek.

SEEK_CUR

Indicates the current position of the file pointer.

SEEK_END

Indicates the end of the file.

SEEK_SET

Indicates the beginning of the file.

See Also

[fseek\(\)](#) — Move file pointer to new location in file

[Iseek\(\) and _Iseek\(\)](#) — Change the position in a file

[File-Related Functions](#) on page 429

SH_*

File-sharing flags

Targets

The `_SH_*` constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The `_SH_*` constants are Extra-ANSI.

Synopsis

```
#include <share.h> /* Required */
```

Description

For files opened by functions `_dos_open()`, `fsopen()`, and `sopen()`, the `_SH_*` constants define the file-sharing mode in which the file is opened.

_SH_COMPAT

Open in compatibility mode. A file opened in this mode can be opened any number of times by any number of processes. This is the default mode, and the apparent mode on DOS, when `share.exe` is not loaded.

_SH_DENYRW

Deny read and write access to the file. This is the exclusive-access mode. A file opened in this mode cannot be opened again until it is closed by the process that set the lock.

_SH_DENYWR

Deny write access to the file. The file can be opened for reading, but requests for write access to the file are denied.

_SH_DENYR

Deny read access to the file. The file can be opened for writing, but requests for read access to the file are denied.

_SH_DENYNO

Deny requests to open the file in compatibility (`_SH_COMPAT`) mode. All other requests are allowed.

See Also

[File-Related Functions](#) on page 429

SHRT_*

Minimum and maximum values for **short** types

Targets

The `SHRT_*` constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The `SHRT_*` constants are ANSI compliant.

Synopsis

```
#include <limits.h> /* Required */
```

Description

The **SHRT_*** constants provide minimum and maximum values for **short** types.

SHRT_MAX

Maximum value for an object of type **short**

SHRT_MIN

Minimum value for an object of type **short**

_S_I*

File permissions

Targets

The **_S_I*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

The **_S_I*** constants are Extra-ANSI.

Synopsis

```
#include <sys/stat.h> /* Required */
```

Description

The **_S_I*** constants indicate whether a file is read-only, write-only, or read/write enabled.

_S_IREAD

Allows the file to be read from later, after it has been closed and then re-opened.

_S_IWRITE

Allows the file to be written to again, after it has been closed and then later reopened.

Do not use this mode alone on DOS or Windows NT. These operating systems do not recognize write-only files, but promote them to read/write files.

Functions that take the **_S_I*** constants as arguments can also take them together as (**_S_IREAD | _S_IWRITE**), which allows both reading and writing.

See Also

[File-Related Functions](#) on page 429

SIG_DFL and SIG_ERR and SIG_IGN

Macros; arguments to **signal()**

Targets

The **SIG_*** macros are supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **SIG_*** macros are ANSI compliant.

Synopsis

```
#include <signal.h> /* Required */
```

Description

The **SIG_*** macros expand to a constant expression of type “pointer to a function returning **void**”, as in:

```
void (*)()
```

On UNIX, the **SIG_*** macros expand to a constant expression of type “pointer to a function taking an **int** argument and returning **void**”, as in:

```
void (*)()(int)
```

This constant expression is distinct from all values obtainable by declaring such a function.

SIG_DFL

Specifies that the given signal is to be handled in the default manner. Use as the second argument to **signal()**.

SIG_ERR

When used as a return value from **signal()**, indicates that **signal()** was unable to redirect the specified signal to the specified function. When used as an argument to **signal()**, returns an error to **raise()** on invoking a signal that has been caught by **SIG_ERR**.

SIG_IGN

Specifies that a given signal is to be ignored. Use as the second argument to **signal()**.

Each **SIG_*** macro expands to an expression that is distinct from the other **SIG_*** macros. See **signal()** for implementation specifics.

See Also

[raise\(\)](#) — [Raise a signal](#)

[signal\(\)](#) — [Set up a signal handler](#)

SIG*

Arguments to **signal()** and **raise()**

Targets

The **SIG*** constants are supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **SIG*** constants are ANSI compliant.

Synopsis

```
#include <signal.h>           /* Required */
```

Description

The **SIG*** constants are positive integer constants. Each one is a signal number corresponding to a particular signal. See **signal()** for implementation specifics.

SIGABRT

Abnormal termination

The signal number corresponding to an abnormal termination. Use as the first argument to **signal()** or **raise()**. On DOS and Windows NT, the default action is to terminate with exit code 3.

SIGFPE

Floating-point error

The signal number corresponding to a floating-point error (erroneous arithmetic operation). Use as the first argument to **signal()** or **raise()**. On DOS and Windows NT, the default action is to terminate with exit code 3.

SIGILL

Invalid instruction

The signal number corresponding to detection of an invalid function image or invalid instruction. Use as the first argument to **signal()** or **raise()**. On DOS and Windows NT, the default action is to terminate with exit code 3.

SIGINT

CONTROL-C or CONTROL-BREAK interrupt

The signal number corresponding to receipt of an interactive attention signal such as CONTROL-C. Use as the first argument to **signal()** or **raise()**. On DOS and Windows NT, the default action is to terminate with exit code 3.

SIGSEGV

Segment violation

The signal number corresponding to an invalid access to a data object. Use as the first argument to **signal()** or **raise()**.

SIGTERM

Termination request

The signal number corresponding to a termination request sent to the program. Use as the first argument to **signal()** or **raise()**, and the second argument to **_kill()**. On DOS and Windows NT, the default action is to terminate with exit code 3.

SIGUSR1**SIGUSR2****SIGUSR3**

User-defined signals

Available for DOS and Windows NT only. Default action is to ignore the signal.

See Also

[raise\(\)](#) — Raise a signal

[signal\(\)](#) — Set up a signal handler

stderr and stdin and stdout

Standard error, input, and output streams

Targets

stderr, stdin, and stdout are supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h> /* Required */
```

Description

Each of these three global variables expands into an expression that points to a FILE-type variable that is initialized at program start-up to control a particular stream.

stderr

Associated with the standard error file, which is initialized as write-only and writes to the screen. By default, stderr is an unbuffered file.

stdin

Associated with the standard input file, which is initialized as read-only and reads from the keyboard. By default, if it references the keyboard, stdin is line-buffered; if it has been redirected, stdin is buffered.

stdout

Associated with the standard output file, which is initialized as write-only and writes to the screen. By default, if it references the screen, stdout is line-buffered; if it has been redirected, stdout is buffered.

The default behaviors can be overridden by calls to [setbuf\(\)](#) or [setvbuf\(\)](#).

See Also

[setbuf\(\)](#) — Specify a buffer for a stream

[setvbuf\(\)](#) — Control file buffering

[typedef struct FILE](#) — Information for controlling a stream

SW_*

Masking constants for 387 status word

Targets

The **SW_*** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

The **SW_*** constants are Extra-ANSI.

Synopsis

```
#include <float.h> /* Required */
```

Description

The **SW_*** masking constants, used with function `_stat387()`, describe the information available in the status word from the 387 coprocessor.

NOTE This information is different from and more complete than the Microsoft floating-point package status word, and is more appropriate to 386/387 operations.

SW_B

Value = 0x8000: B unmasked exception flag

SW_C0

Value = 0x0100: Condition bit 0

SW_C1

Value = 0x0200: Condition bit 1

SW_C2

Value = 0x0400: Condition bit 2

SW_C3

Value = 0x4000: Condition bit 3

SW_DENORMAL

Value = 0x0002: Denormalized operand

SW_INEXACT

Value = 0x0020: Precision (inexact result)

SW_INVALID

Value = 0x0001: Invalid operation

SW_IR

Value = 0x0080: IR unmasked exception flag

SW_OVERFLOW

Value = 0x0008: Overflow

SW_STACKFAULT

Value = 0x0040: Floating-point stack fault

SW_TOS

Value = 0x3800: FP stack pointer (3 bits, 0 — 7)

SW_UNDERFLOW

Value = 0x0010: Underflow

SW_ZERODIVIDE

Value = 0x0004: Zero divide

Note These 387 floating-point status-word **SW_*** masking constants are to be used only with function `_stat387()`.

See Also

_threadid

ID of currently running thread

Targets

Variable `_threadid` is supported on the following target platforms:

EMB	UNIX	DOS	NT
—	—	—	x

ANSI

Extra-ANSI

Synopsis

```
#include <process.h>
extern long _threadid;
```

Description

`_threadid` is the thread ID of the currently running thread, represented as a **long** integer.

__TIME__

File time of translation

Targets

Predefined macro `__TIME__` is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

```
--TIME__ /* Predefined macro */
```

Description

--TIME__ produces the time of translation of the source file as a string of this form:

hh:mm:ss

hh is in the range 01 - 24 hours; *mm* is in the range 00 - 59 minutes; and *ss* is in the range 00 - 59 seconds.

The value of this macro remains invariant across a single translation unit.

See Also

[DATE](#) — [File date of translation](#)

[Time-Related Functions](#) on page 434

timezone

Time zone; hours from UTC, expressed in seconds

Targets

Variable `timezone` is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <time.h> /* Required */
```

Description

Function `tzset()` or `_tzset()` converts the number of hours from UTC (Coordinated Universal Time) to seconds and places the result in the global variable `timezone`.

See Also

[ctime\(\) and _ctime\(\)](#) — [Convert a time_t value to printable form](#)

[gmtime\(\) and _gmtime\(\)](#) — [Convert a time_t value to a struct tm, adjusted to UTC](#)

[localtime\(\) and _localtime\(\)](#) — [Convert a time_t value to a struct tm](#)

[time\(\)](#) — [Get the current time and date](#)

[tzset\(\) and _tzset\(\)](#) — [Set local time-zone values](#)

TMP_MAX

Minimum number of unique file names generated by `tmpnam()`

Targets

Constant **TMP_MAX** is supported on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h> /* Required */
```

Description

The integer constant **TMP_MAX** specifies the number of file names that **tmpnam()** must generate before repeating.

U*_MAX

Maximum values for **unsigned** types

Targets

The **U*_MAX** constants are defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

The **U*_MAX** constants are ANSI compliant.

Synopsis

```
#include <limits.h> /* Required */
```

Description

The **U*_MAX** constants provide maximum values for **unsigned** types.

UCHAR_MAX

Maximum value for an object of type **unsigned char**

UINT_MAX

Maximum value for an object of type **unsigned int**

ULONG_MAX

Maximum value for an object of type **unsigned long**

USHRT_MAX

Maximum value for an object of type **unsigned short**

Chapter 6 — Typedefs and Structs

In This Chapter

- [Listing of Typedefs, Structs, and Unions](#)

Listing of Typedefs, Structs, and Unions

This section lists the typedefs, structs, and unions in alphabetical order.

BYTEREGS

Contents of 386 family hardware byte registers

Targets

Structure **BYTEREGS** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h> /* Required */
#include <bios.h>
struct BYTEREGS {
    unsigned char al, ah, xax[sizeof(int)-2];
    unsigned char bl, bh, xbx[sizeof(int)-2];
    unsigned char cl, ch, xcx[sizeof(int)-2];
    unsigned char dl, dh, xdx[sizeof(int)-2];
};
```

Description

The **BYTEREGS struct** stores information about the hardware byte registers on 32-bit 386 family processors.

- *al, bl, cl, and dl* are low-order hardware byte registers.
- *ah, bh, ch, and dh* are high-order hardware byte registers.
- *xax[], xbx[], xcx[], and xdx[]* are dummy filler space constituting two more bytes.

See Also

union [REGS](#) — [Contents of 386 family word- and byte-register structures](#)

clock_t

Representation of elapsed processor time

Targets

Type **clock_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <time.h> /* Required */
```

Description

`clock_t` is an arithmetic type that represents elapsed processor time.

See Also

[clock\(\)](#) — Report elapsed processor time

[time\(\)](#) — Get the current time and date

Constant [CLOCKS_PER_SEC](#) — Clock ticks converted to seconds

complex

A complex number

Targets

Structure `complex` is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Non-ANSI

Synopsis

```
#include <math.h> /* Required */
struct complex {
    double x, y;
};
```

Description

The `complex struct` represents a complex number, where `x` is the real part and `y` is the imaginary part.

See Also

[cabs\(\) and _cabs\(\)](#) — Compute complex absolute value

_dirent

A POSIX file name for `_readdir()`

Targets

Structure `_dirent` is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Synopsis

```
#include <dirent.h> /* Required */
/* For Windows NT targets */
struct _dirent {
```

diskfree_t

```
char d_name[256];
} ;
/* For DOS targets */
struct _dirent {
    char d_name[13];
} ;
```

Description

The **_dirent** struct stores the name of a file to be read by **_readdir()**.

For Windows NT targets, the file name is long (up to 255 characters plus terminating null character).

For DOS targets, the file name is the standard 12 characters plus the terminating null character (*d:filename.ext*), where *d* is the single drive letter a, b, and so on; *filename* is 1 to 8 characters; and *ext* is 0 (zero) to 3 characters.

See Also

[_readdir\(\)](#) — Read a POSIX directory stream

diskfree_t

Information about space on a disk drive

Targets

Structure **diskfree_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h> /* Required */

/* For Windows NT targets */
struct diskfree_t{
    unsigned total_clusters;
    unsigned avail_clusters;
    unsigned sectors_per_cluster;
    unsigned bytes_per_sector;
};

/* For DOS targets */
struct diskfree_t{
    unsigned short total_clusters;
    unsigned short avail_clusters;
    unsigned short sectors_per_cluster;
    unsigned short bytes_per_sector;
};
```

Description

The **diskfree_t** struct stores information about storage space on a disk drive.

- *total_clusters* is the total number of clusters available on the disk.
- *avail_clusters* is the number of free clusters available on the disk.

- *sectors_per_cluster* is the number of sectors per cluster.
- *bytes_per_sector* is the number of bytes per sector.

div_t

Result type of function **div()**

Targets

Structure **div_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdlib.h> /* Required */
typedef struct {
    int quot;
    int rem;
} div_t;
```

Description

The **div_t** struct is the type returned from **div()**. Structure member *quot* is the quotient; *rem* is the remainder.

See Also

[div\(\)](#) — Perform integer division with remainder

dosdate_t

Information about current DOS system date

Targets

Structure **dosdate_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h> /* Required */
struct dosdate_t{
    unsigned char day;
    unsigned char month;
    unsigned int year;
    unsigned char dayofweek;
};
```

Description

The **dosdate_t** struct stores information about a date for the DOS date functions. The structure members have the following value ranges:

- *day* is the day of the month (1 - 31).
- *month* is the month of the year (1 - 12; January == 1).
- *year* is the Gregorian calendar year (1980 - 2099).
- *dayofweek* is the number of days since Sunday (0 - 6).

DOSERROR

Detailed information about a DOS error

Targets

Structure **DOSERROR** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	—

ANSI

Non-ANSI

Synopsis

```
#include <dos.h> /* Required */
struct DOSERROR{
    short int exterror;
    char class;
    char action;
    char locus;
} ;
```

Description

The **DOSERROR** struct stores detailed information about a DOS run-time error. The structure members have the following meanings:

- *exterror* is the extended error code; contents of ax register.
- *class* is the error class; contents of bh register.
- *action* is the recommended action; contents of bl register.
- *locus* is the error locus; contents of ch register.

dostime_t

Information about current system time

Targets

Structure **dostime_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h> /* Required */
struct dostime_t{
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
    unsigned char hsecond;
};
```

Description

The **dostime_t struct** stores the current system time. The structure members have the following meanings and ranges:

- *hour* is the number of hours since midnight (0 - 23).
- *minute* is the number of minutes since the start of the hour (0 - 59).
- *second* is the number of seconds since the start of the minute (0 - 59).
- *hsecond* is the number of elapsed hundredths of a second (0 - 99).

DWORDREGS

Contents of 386 family hardware word registers

Targets

Structure **DWORDREGS** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h> /* Required */
struct DWORDREGS {
    unsigned int eax ;
    unsigned int ebx ;
    unsigned int ecx ;
    unsigned int edx ;
    unsigned int esi ;
    unsigned int edi ;
    unsigned int cflag ;
};
```

Description

The **DWORDREGS struct** stores information about the hardware general registers on the 32-bit 386 family of processors.

The names of the structure members correspond to the names of the registers whose content values are stored in (or will be loaded from) each member. For example, structure member *eax* contains a value identical to the value stored in (or to be loaded into) hardware register %eax.

See Also

union REGS — [Contents of 386 family word- and byte-register structures](#)

struct WORDREGS — [Contents of 386 family hardware word registers](#)

exception

Nature of a math error

Targets

Structure **exception** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	X	X

ANSI

Non-ANSI

Synopsis

```
#include <math.h> /* Required */
struct exception {
    int type;
    char *name;
    double arg1;
    double arg2;
    double retval;
};
```

Description

The **exception struct** is passed to the **matherr()** function when a floating-point exception is detected. The structure members have the following meanings and possible values:

- *type* contains one of the following values specifying the exception type:
 - DOMAIN** is an argument domain error, as in **sqrt(-1e0)**.
 - SING** is an argument singularity, as in **pow(0e0,-2)**.
 - OVERFLOW** is an overflow range error, as in **pow(10e0,100)**.
 - UNDERFLOW** is an underflow range error, as in **pow(10e0,-100)**.
 - TLOSS** is a total loss of precision, as in **exp(1000)**.
 - PLOSS** is a partial loss of precision, as in **sin(10e70)**
- *name* points to a string containing the name of the function that detected the error.
- *arg1* and *arg2* give the values that caused the error (if required).
- *retval* contains the value returned by the math function that caused the error.

System issues

For DOS targets, the possible values for the *type* field are **DOMAIN**, **SING**, **OVERFLOW**, **UNDERFLOW**, **TLOSS**, and **PLOSS** (without leading underscores), respectively.

See Also

[matherr\(\), matherrx\(\) and _matherr\(\), _matherrx\(\)](#) — Provide standard error reporting for math functions

[set_matherr\(\)](#) — User defined math error function

FILE

Information for controlling a stream

Targets

Structure **FILE** is defined for the following target platforms:

EMB	UNIX	DOS	NT
X	X	X	X

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h> /* Required */
typedef struct {
    /*
     * See stdio.h for details
     */
} FILE ;
```

Description

FILE is a **struct** that contains the information required to control a stream. A variable of type **FILE *** is commonly used to designate a particular stream.

Note The definition of the **FILE struct** differs for each target platform. For details about the definition of **FILE** for your target platform, see the **stdio.h** header file on your distribution.

The **FILE struct** contains some or all of the following members, depending on the target platform.

- *_cnt* specifies a number of characters or bytes.
 - If *_IOREAD* is defined, *_cnt* specifies the number of characters remaining to be read in the buffer.
 - If *_IOWRT* is defined, *_cnt* specifies the number of unused bytes remaining in the buffer.
- *_ptr* specifies a byte location.
 - If *_IOREAD* is defined, *_ptr* specifies the location of the next byte to be read.
 - If *_IOWRT* is defined, *_ptr* specifies the location of the next unused byte.
- *_base* points to the beginning of the buffer. This value remains constant after the file is opened.
- *_bufsiz* specifies the size of the buffer.
- *_bufendp* points to the byte following the buffer.
- *_bufendtab[]* is a table of *_bufendp* pointers.
- *_flag* keeps track of state (EOF, error, and whether the file is opened for reading, writing, or both).

- *_file* contains the low-level system file handle.
- *_unused[]* is an array of **char** or **int** types, included for alignment purposes (in case pragma **Align_members** is off).

See Also[File-Related Functions](#)

finddata_t

Attributes of a file

Targets

Structure **finddata_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Extra-ANSI

Synopsis

```
#include <io.h> /* Required */
struct finddata_t {
    unsigned attrib;
    time_t    time_create;
    time_t    time_access;
    time_t    time_write;
    _fsize_t size;
    char      name[260];
};
```

Description

The **_find***(*)* functions use the **finddata_t** structure to specify file information. The structure members have the following meanings:

- *attrib* is the file attribute.
Possible values are [**A_ARCH**](#), [**A_HIDDEN**](#), [**A_NORMAL**](#), [**A_RDONLY**](#), [**A_SYSTEM**](#), [**A_SUBDIR**](#), and [**A_VALID**](#).
- *time_create* is a structure that contains time of creation (-1L for file allocation systems).
- *time_access* is a structure that contains time of last file access (-1L for file allocation systems).
- *time_write* is a structure that contains time of last write to file.
- *size* is the file length in bytes.
- *name* is the file name.

See Also

[**findclose\(\)** and **findfirst\(\)** and **findnext\(\)**](#), [**findclose\(\)** and **findfirst\(\)** and **findnext\(\)**](#), [**findclose\(\)** and **findfirst\(\)** and **findnext\(\)**](#) — Find a file whose name matches a specification

Type [**time_t**](#) — [Representation of a date and time](#)

Constants [**A_***](#) — [File attributes](#)

find_t

Name and attributes of a file

Targets

Structure **find_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h>      /* Required */

/* For Windows NT targets */
struct find_t {
    long             reserved;
    unsigned long    attrib;
    unsigned short   wr_time;
    unsigned short   wr_date;
    long int         size;
    char             name[256];
};

/* For DOS targets */
struct find_t{
    char             reserved[21];
    char             attrib;
    unsigned short   wr_time;
    unsigned short   wr_date;
    long int         size;
    char             name[13];
} ;
```

Description

- The **_dos_find***() functions use the **find_t** structure to specify file information.
- reserved* is reserved by DOS.
- attrib* is an attribute byte for matched file specification. It is composed by ORing together a combination of the following constants: [A_ARCH](#), [A_HIDDEN](#), [A_NORMAL](#), [A_RDONLY](#), [A_SYSTEM](#), [A_SUBDIR](#), and [A_VALID](#).
- wr_time* is the time of the last write to the file.
- wr_date* is the date of the last write to the file.
- size* is the byte length of the file.
- name* is the name of the matched file or directory, including the terminating NUL character ('\0').

System issues

For Windows NT targets, the file name is long — up to 255 characters plus the terminating NUL character ('\0').

fpos_t

For DOS targets, the file name is the standard 12 characters plus the terminating NUL character ('\0')—*d:filename.ext*, where *d* is the single drive letter a, b, and so on; *filename* is 1 to 8 characters; and *ext* is 0 (zero) to 3 characters.

See Also

Constants [A-* — File attributes](#)

fpos_t

Unique file-position type

Targets

Type **fpos_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdio.h> /* Required */
```

Description

fpos_t is a type suitable for containing a unique file position as obtained from **fgetpos()**.

See Also

[fgetpos\(\) — Determine file position](#)

[fsetpos\(\) — Set file position](#)

_HEAPINFO

Information about the heap

Targets

Structure **_HEAPINFO** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

Extra-ANSI

Synopsis

```
#include <malloc.h> /* Required */
typedef struct{
    void    *_pentry;
    size_t   _size;
    int     _useflag;
} _HEAPINFO
```

Description

`_heapwalk()` uses the `_HEAPINFO` structure to walk through the heap, one entry at a time. The structure members have the following meanings:

- `_pentry` is a pointer to the heap entry.
- `_size` is the size of the heap entry.
- `_useflag` is the “Heap entry in use” flag.

Upon return from the first call to `_heapwalk()`, which sets `_pentry` to `NULL`, the fields are set as follows:

- `_pentry` points to the first block of the heap.
- `_size` is set to the size of the first heap entry.
- `_useflag` is set to `_FREEENTRY` or `_USEDENTRY`.

See Also

[heapwalk\(\) — Walk through the heap](#)

Constants [HEAP* — State of the heap](#)

jmp_buf

Information to restore an environment

Targets

Type `jmp_buf` is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <setjmp.h> /* Required */
```

Description

`jmp_buf` is an array type that can hold the information needed to restore a calling environment.

See Also

[longjmp\(\) — Execute a non-local jump](#)

[setjmp\(\) — Save a reference to the current calling environment for a subsequent non-local jump](#)

lconv

Locale-specific numeric representation

Targets

Structure `lconv` is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <locale.h> /* Required */
struct lconv {
    char *decimal_point;
    char *thousands_sep;
    char *grouping;
    char *int_curr_symbol;
    char *currency_symbol;
    char *mon_decimal_point;
    char *mon_thousands_sep;
    char *mon_grouping;
    char *positive_sign;
    char *negative_sign;
    char int_frac_digits;
    char frac_digits;
    char p_cs_precedes;
    char p_sep_by_space;
    char n_cs_precedes;
    char n_sep_by_space;
    char p_sign_posn;
    char n_sign_posn;
};
```

Description

The **lconv** struct describes the way numbers are represented for a locale.

NOTE For the ANSI Standard C locale, no values are available except *decimal_point*, which is set to the '.' character.

The structure members have the following meanings:

- *decimal_point* is a decimal-point character used with non-monetary quantities.
- *thousands_sep* is a separator for groups of digits to the left of the decimal-point character in non-monetary quantities.
- *grouping* is a string indicating the size of each group of digits in non-monetary quantities.
- *int_curr_symbol* is the international currency symbol, left-justified in a four-character field.
- *currency_symbol* is a local currency symbol.
- *mon_decimal_point* is a decimal point used with monetary quantities.
- *mon_thousands_sep* is a separator for groups of digits to the left of the decimal-point character in monetary quantities.
- *mon_grouping* is a string indicating the size of each group of digits in monetary quantities.
- *positive_sign* is a string used to indicate a non-negative monetary quantity.
- *negative_sign* is a string used to indicate a negative monetary quantity.

- *int_frac_digits* is the number of fractional digits to be displayed in an internationally formatted monetary quantity.
- *frac_digits* is the number of fractional digits to be displayed in a monetary quantity.
- *p_cs_precedes* is set to 1 (one) if the currency symbol precedes the value of a non-negative monetary quantity. It is set to 0 (zero) if the currency symbol follows the value of a non-negative monetary quantity.
- *p_sep_by_space* is set to 1 (one) if the currency symbol is separated by a space from the value of a non-negative monetary quantity. It is set to 0 (zero) if the currency symbol is not separated by a space from the value of a non-negative monetary quantity.
- *n_cs_precedes* is set to 1 (one) if the currency symbol precedes the value of a negative monetary quantity. It is set to 0 (zero) if the currency symbol follows the value of a negative monetary quantity.
- *n_sep_by_space* is set to 1 (one) if the currency symbol is separated by a space from the value of a negative monetary quantity. It is set to 0 (zero) if the currency symbol is not separated by a space from the value of a negative monetary quantity.
- *p_sign_posn* is the position of the *positive_sign* in a non-negative monetary quantity.
- *n_sign_posn* is the position of the *negative_sign* in a negative monetary quantity.

NOTE If a structure member of type **char** * has the value "", or a member of type **char** has the value CHAR_MAX, that value is not available in the current locale.

The strings for *grouping* and *mon_grouping* are sequences of elements interpreted as follows:

- 0 means repeat previous element for remainder of digits.
- MAX_CHAR means no further grouping.
- Any other value means the number of digits in the current group. Each next element corresponds to each next group of digits to the left of the current group.

The values of *p_sign_posn* and *n_sign_posn* are interpreted as follows:

- 0 — Parentheses surround the quantity and the *currency_symbol*.
- 1 — The sign string precedes the quantity and the *currency_symbol*.
- 2 — The sign string follows the quantity and the *currency_symbol*.
- 3 — The sign string immediately precedes the *currency_symbol*.
- 4 — The sign string immediately follows the *currency_symbol*.

See Also

[localeconv\(\)](#) — Query current locale for numeric format

[setlocale\(\)](#) — Set the current locale

ldiv_t

Result type of function **ldiv()**

Targets

Structure **ldiv_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stdlib.h> /* Required */
typedef struct {
    long quot;
    long rem;
} ldiv_t;
```

Description

Structured type returned from **ldiv()**. Structure member *quot* is the quotient; *rem* is the remainder.

See Also

[ldiv\(\)](#) — Perform long int division with remainder

ptrdiff_t

Integral type; difference of two pointers

Targets

Type **ptrdiff_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stddef.h> /* Required */
```

Description

ptrdiff_t is a macro that expands to the integral type of the result of subtracting two pointers.

REGS

Contents of 386 family word- and byte-register structures

Targets

Union **REGS** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h>
union REGS {
    struct WORDREGS x;
    struct BYTEREGS h;
    struct BYTEREGS l;
};
```

Description

The **REGS** union stores information about the contents of the hardware registers on the 386 family of processors.

See Also

struct BYTEREGS — [Contents of 386 family hardware byte registers](#)

struct WORDREGS — [Contents of 386 family hardware word registers](#)

sig_atomic_t

Integral type; atomic entity

Targets

Type **sig_atomic_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <signal.h>           /* Required */
```

Description

sig_atomic_t is an integral type of object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.

size_t

Type of the result of **sizeof()**

Targets

size_t is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stddef.h>           /* Required */
```

Description

size_t is a type of the result of the **sizeof()** operator.

size_t is defined in **sizet.h**, which is #included in **stddef.h**, **stdio.h**, **stdlib.h**, **string.h**, and **time.h**.

SREGS

Contents of 386 family segment registers

Targets

Structure **SREGS** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h>    /* One of these */
#include <bios.h>    /* is required */
struct SREGS {
    unsigned short int es,
    unsigned short int cs,
    unsigned short int ss,
    unsigned short int ds
} ;
```

Description

The **SREGS struct** describes the contents of the 386 family segment registers. The names of the structure members correspond to the names of the segment registers whose content values are stored in (or will be loaded from) each member. For example, structure member *es* contains a value identical to the value stored in (or to be loaded into) segment register %es.

stat

Information about a file

Targets

Structure **stat** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <sys\stat.h>    /* Required */
struct stat {
    dev_t st_dev;
    ino_t st_ino;
    unsigned short st_mode;
    short st_nlink;
```

```
short st_uid;
short st_gid;
dev_t st_rdev;
off_t st_size;
time_t st_atime;
time_t st_mtime;
time_t st_ctime;
} ;
```

Description

The **stat struct**, used by functions **fstat()** and **stat()**, stores information about a file. The structure members have the following meanings:

- *st_dev* is the drive number or device handle (0 is drive A, 1 is drive B, and so on)
- *st_ino* is the inode number of the file (see **System issues**)
- *st_mode*, the file-mode information, is one of the following constants:

_S_IFDIR means *path* refers to a directory.

_S_IFREG means *path* refers to an ordinary file.

_S_IREAD means the file has read permission.

_S_IWRITE means the file has write permission.

_S_IEXEC means the file is executable (determined by the file-name extension).

- *st_nlink* is the number of hard links; always defined as 1 (one).
- *st_uid* is the User ID, always root (see **System issues**).
- *st_gid* is the Group ID, always root (see **System issues**).
- *st_rdev* is the drive number (same as *st_dev*).
- *st_size* is the length of the file in bytes.
- *st_atime* is the time of the last file access; type **time_t**.
- *st_mtime* is the time of the last file modification; type **time_t**.
- *st_ctime* is the time of the last file status change; type **time_t**.

System issues

On DOS targets, the *st_ino*, *st_uid*, and *st_gid* structure members do not contain meaningful values.

See Also

[stat\(\) and stat\(\)](#) — Get information about a file or directory

Type [time_t](#) — Representation of a date and time

timeb

Time-zone information from function **ftime()**

Targets

Structure **timeb** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <sys\timeb.h> /* Required */
struct timeb {
    time_t          time;
    unsigned short  millitm;
    short           timezone;
    short           dstflag;
};
```

Description

The **timeb** struct contains time-zone information stored in it by function [ftime\(\) and _ftime\(\)](#). The structure members have the following meanings:

- *time* is the number of seconds since 00:00:00 January 1, 1970 in Coordinated Universal Time.
- *millitm* is the number of milliseconds between the value of the *time* field and the current time (useful as a means of getting the current time with millisecond granularity).
- *timezone* is the number of minutes west of Coordinated Universal Time to the current time zone. (Might be negative to indicate east of Coordinated Universal Time.)
- *dstflag* is the Daylight Saving Time flag (Boolean).
 - 1 (positive) if Daylight Saving Time is in effect.
 - 0 (zero) if Daylight Saving Time is not in effect.
 - 1 (negative) if the Daylight Saving Time information is not available.

See Also

[ftime\(\) and _ftime\(\)](#) — [Get current time values](#)

[Time-Related Functions](#) on page 434

Type [time_t](#) — [Representation of a date and time](#)

time_t

Representation of a date and time

Targets

Type **time_t** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <time.h>          /* Required */
```

Description

time_t is an arithmetic type that can represent time and date.

See Also

[ctime\(\) and _ctime\(\)](#) — Convert a **time_t** value to printable form

[difftime\(\)](#) — Calculate difference between two times

[gmtime\(\) and _gmtime\(\)](#) — Convert a **time_t** value to a struct **tm**, adjusted to UTC

[localtime\(\) and _localtime\(\)](#) — Convert a **time_t** value to a struct **tm**

[mktime\(\)](#) — Convert a struct **tm** to a **time_t** value

[time\(\)](#) — Get the current time and date

[Time-Related Functions](#) on page 434

[struct tm](#) — Components of a date and time

[struct utimbuf](#) — Information about file access and modification times

tm

Components of a date and time

Targets

Structure **tm** is defined for the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <time.h>      /* Required */
struct tm {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
};
```

Description

The **tm** struct holds individual **int** components of time and date. The structure members have the following normal ranges:

- *tm_sec* is the number of seconds after the minute (0 - 59).
- *T-man* is the number of minutes after the hour (0 - 59).
- *tm_hour* is the number of hours since midnight (0 - 23).
- *tm_mday* is the number of the day of the month (1 - 31).
- *tm_mon* is the number of months since January (0 - 11).
- *tm_year* is the number of years since 1900 (0 - *Infinity*).
- *tm_wday* is the number of days since Sunday (0 - 6).
- *tm_yday* is the number of days since January 1 (0 - 365).
- *tm_isdst* is the Daylight Saving Time flag (Boolean).
1 (positive) if Daylight Saving Time is in effect.
0 (zero) if Daylight Saving Time is not in effect.
-1 (negative) if the Daylight Saving Time information is not available.

See Also

[asctime\(\) and _asctime\(\)](#) — Convert a struct tm to printable form

[gmtime\(\) and _gmtime\(\)](#) — Convert a time_t value to a struct tm, adjusted to UTC

[localtime\(\) and _localtime\(\)](#) — Convert a time_t value to a struct tm

[mktime\(\)](#) — Convert a struct tm to a time_t value

[Time-Related Functions](#) on page 434

utimbuf

Information about file access and modification times

Targets

Structure **utimbuf** is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <utime.h> /* Required */
struct utimbuf {
    time_t actime;
    time_t modtime;
};
```

Description

The **_utimbuf** struct stores information about the time of the most recent access to a file and the latest modification to that file.

- *actime* is the last time the file was accessed.
- *modtime* is the last time the file was modified.

See Also[utime\(\) and _utime\(\)](#) — Update a file's date/time stamp[Time-Related Functions](#) on page 434Type [time_t](#) — Representation of a date and time

va_list

Information needed by var-arg macros

TargetsType **va_list** works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Reentrancy

Reentrant

Synopsis

#include <stdarg.h> /* Required */

Description

va_list expands to a type suitable for holding the information needed by the var-arg (variable-argument) macros. The function using the macros must declare a variable of this type.

See Also[va_arg\(\)](#) — Get the next argument in a variable series of arguments[va_end\(\)](#) — Terminate va_list() processing[va_start\(\)](#) — Initialize an object of type va_list[vbprintf\(\)](#) — Print to a string of given size using var-arg macros[vfprintf\(\)](#) — Print to a file using var-arg macros[vprintf\(\)](#) — Print to `stdout` using var-arg macros[vsprintf\(\)](#) — Print to a string using var-arg macros[vfscanf\(\)](#) and [_vfscanf\(\)](#) — Read values from a file using var-arg macros[vscanf\(\)](#) and [_vscanf\(\)](#) — Read values from `stdin` using var-arg macros[vsscanf\(\)](#) and [_vsscanf\(\)](#) — Read values from a string using var-arg macros

wchar_t

Integral type; extended-character-set values

Targets

Type `wchar_t` works on the following target platforms:

EMB	UNIX	DOS	NT
x	x	x	x

ANSI

ANSI compliant

Synopsis

```
#include <stddef.h> /* Required */
```

Description

`wchar_t` is an integral type with range suitable for representation of all values of the largest extended character set of all supported locales.

See Also

[mblen\(\)](#) — Determine length of a multibyte character

[mbstowcs\(\)](#) — Convert a multibyte string to `wchar_t` array

[mbtowc\(\)](#) — Convert a multibyte character to `wchar_t` code

[wcstombs\(\)](#) — Convert a `wchar_t` array to a multibyte string

[wctomb\(\)](#) — Convert a `wchar_t` code to a multibyte character

Constant [MB_LEN_MAX](#) — Maximum number of bytes in a multibyte character

WORDREGS

Contents of 386 family hardware word registers

Targets

Structure `WORDREGS` is defined for the following target platforms:

EMB	UNIX	DOS	NT
—	—	x	x

ANSI

Non-ANSI

Synopsis

```
#include <dos.h> /* Required */
struct WORDREGS {
    unsigned int ax ;
    unsigned int bx ;
    unsigned int cx ;
    unsigned int dx ;
    unsigned int si ;
    unsigned int di ;
    unsigned int cflag ;
};
```

Description

The **WORDREGS struct** describes the contents of the 32-bit 386 family hardware word registers. **WORDREGS** is provided for compatibility with previous source code. **WORDREGS** and **DWORDREGS** have the same size arguments.

The names of the structure members correspond to the names of the registers whose content values are stored in (or will be loaded from) each member. For example, structure member *ax* contains a value identical to the value stored in (or to be loaded into) hardware register %ax.

See Also

struct [DWORDREGS](#) — [Contents of 386 family hardware word registers](#)

Appendix A — Input and Output

In This Appendix

- [Logical Data Streams](#)
- [Opening and Closing Files](#)
- [Text and Binary Streams](#)
- [Errors](#)
- [Flushing Buffers](#)
- [End of File](#)

Logical Data Streams

Input and output can be directed from and to physical devices: the console, disk drives, tape drives, printers, and so on. To provide a consistent interface, input and output are mapped to logical data streams.

A *stream* is an ordered sequence of bytes. These streams are associated with files, which might correspond to external storage (such as a disk file) or might not (such as the console or printer).

Buffered Streams

A stream can be *buffered*, which means that bytes are typically read and written as follows:

- read in standard amounts from an external device into a buffer
- read in arbitrary amounts from the buffer to objects in a program
- written in standard amounts to an external device out of a buffer
- written in arbitrary amounts to the buffer from objects in a program

The buffer might be designated as an object of the program or it might be automatically allocated by the library.

File Pointers

If a file can support positioning requests, a file pointer associated with a stream maintains a record of the current position in the file through any read, write, and positioning requests.

If a file cannot support positioning requests, all input and output is done at the end of the stream.

Aside from positioning, logical streams hide any differences among the various types of input and output devices.

Opening and Closing Files

Opening a file associates a stream with the file, and positions the file pointer at the beginning of the file (if the file can support a file pointer). It may involve creating the file, which causes any previous contents to be deleted.

At program start-up, three files are opened automatically (see [stderr and stdin and stdout](#), [stderr and stdin and stdout](#), and [stderr and stdin and stdout](#)).

Closing a file disassociates the stream from the file. If an output stream is buffered, the buffer is flushed into the stream when the file is closed. Subsequently, the file may be reopened.

All files are closed if the function **main()** returns or if the function **exit()** is executed.

Text and Binary Streams

A stream can be designated as a text stream or a binary stream.

- A *text stream* is assumed to represent characters. If the external representation of a character sequence does not match the C language representation of that sequence, transformations might be performed upon the bytes making up a text stream.
- A *binary stream* is not subjected to transformations upon input or output. A sequence of bytes stored on an external device will exactly match a program's internal storage of that sequence of bytes, after a read or write.

NOTE On most UNIX systems, text and binary streams behave identically.

Errors

Some I/O functions return values that make it possible to distinguish failure from success. For example, **fputc()** returns its input value if it succeeds in putting the character in the stream, and returns EOF if it fails.

When a function writes to a buffered output stream, a return value denoting success means that it has successfully written either to the external storage or to the stream's buffer.

CAUTION A function that writes to a buffered output stream can return information to the effect that it has successfully written characters, but a subsequent write error might prevent those characters from being written to external storage.

If a write error does occur, **errno** and the stream's error flag are set. The one case where it is impossible to return information about write failures is if the failure occurs when a stream's buffer is being flushed at program termination.

Variable **errno**

Several functions reference the **int** variable **errno**, which is declared in **errno.h** and is set to 0 (zero) at program start-up. Some library functions set **errno** to a positive integer error code when an error occurs during their execution. However, no library function ever sets **errno** to 0 (zero).

To reference **errno** from a program, the header file **errno.h** must be included. For more information about global variable **errno**, see [Constants and Globals](#) on page 319.

Error Flag

An error flag is associated with each stream. If a read error or write error has occurred on the stream, the error flag is set. The end-of-file and error flags are *sticky*. That is, they are not cleared by subsequent calls to library functions, except **clearerr()**, **fseek()**, and **rewind()**. These functions specifically include clearing end-of-file and error flags.

Flushing Buffers

If an application requires certainty about the success of a write, the program should call **fflush()** to explicitly flush the buffer. If no errors have been reported previously and **fflush()** returns a value denoting success, all output has been successfully written to external storage. That is, all output has been accepted by the operating system for writing.

However, external events could still cause the operating system to fail to perform the actual physical I/O.

End of File

End-of-File Flag

An end-of-file flag is associated with each stream. The end-of-file flag is set upon a read request when a file pointer is already positioned at the end of the file. Thus it is not always the case that when the file pointer is positioned at the end of a file, the end-of-file flag has been set for the stream.

If the end-of-file flag is set for a stream (see **feof()**), no characters are read.

End of File on DOS and Windows NT Targets

Many editors and system utilities treat the character CTRL-Z (ASCII 26) as an end-of-file indicator. A CTRL-Z encountered in an input text stream indicates end-of-file, and no further characters are read from the stream.

The CTRL-Z itself is ignored in the sense that it is read by library input functions, but is not returned by them — **fgetc(F)** returns EOF, if F is a text stream whose file pointer points at CTRL-Z. CTRL-Z is not treated in any special manner for binary streams.

Appendix B — Unicode Functions

In This Appendix

- [MetaWare C Unicode Functions](#)
- [Using Unicode Functions](#)
- [Unicode Function in assert.h](#)
- [Unicode Functions in ctype.h](#)
- [Unicode Functions in stdio.h](#)
- [Unicode Functions in stdlib.h](#)
- [Unicode Functions in string.h](#)
- [Unicode Functions in time.h](#)

MetaWare C Unicode Functions

The MetaWare C Run-Time Library provides the Unicode functions discussed in this chapter for some target systems.

NOTE All the functions described in this appendix are Extra-ANSI.

What is Unicode?

Unicode is a character-encoding standard that is meant to eventually replace the ASCII standard character encoding. ASCII is an eight-bit character encoding, while Unicode is a 16-bit encoding.

Are MetaWare C Unicode Functions Available on my Target System?

The libraries shipped with the MetaWare C compiler are configured so that the Unicode routines that need to pass Unicode file-name strings to the operating system first translate the Unicode string into ASCII, then call the standard operating system routine.

You can use the Unicode functionality of the MetaWare C compiler and libraries only if the size of type **wchar_t** is 2, as reported by the compiler. The following program reports the size of type **wchar_t**:

```
#include <stddef.h>
#include <stdio.h>
main()
{
    printf("sizeof(wchar_t) = %d\n", sizeof(wchar_t));
}
```

If the Size of wchar_t is 2

For most targets, if the size of type **wchar_t** is 2 as shipped, the MetaWare C libraries contain the Unicode support functions ready to use. In these cases, all you need to do is modify the library source file **fio_g/_uopen.c** to call the specific operating-system interface routines that support Unicode filenames.

The version of **_uopen.c** that ships with the MetaWare C run-time library source code translates Unicode file names to ASCII and calls the standard non-Unicode operating-system interface. This file can be modified to adapt to systems that have true Unicode support, to avoid the translation of file names to ASCII.

If the Size of wchar_t is not 2

If the size of **wchar_t** is not 2 by default on your target system, you must do the following to use Unicode routines in your code:

- Recompile the entire MetaWare C run-time library with driver option **-Hwesize=2** specified.
- Modify the file **fio_g/_uopen.c** to call the specific operating-system interface routines that support Unicode file names on your target.
- Recompile all application code using Unicode with driver option **-Hwesize=2** specified.

NOTE If your MetaWare C distribution does not contain run-time library source code, or if the file **fio_g/_uopen.c** is not present, contact your vendor for information about licensing library source code.

Using Unicode Functions

The header files `stdio.h`, `stdlib.h`, `string.h`, and `time.h` define the type `_UNICHAR`, the type of a Unicode character (equivalent to `wchar_t`). This type is defined only if the macro `_UNICODE_` has been previously defined. These header files also define the macro `_unichar`, which is equivalent to `_UNICHAR`.

The tables in the following sections list the Unicode function equivalents for standard (ASCII) functions, the corresponding dual-mode macros, and brief descriptions of what the functions do.

- [Unicode Function in assert.h](#) on page 392
- [Unicode Functions in ctype.h](#) on page 392
- [Unicode Functions in stdio.h](#) on page 393
- [Unicode Functions in stdlib.h](#) on page 394
- [Unicode Functions in string.h](#) on page 395
- [Unicode Functions in time.h](#) on page 396

Most Unicode functions in the header files are identical to existing ASCII functions, except that `char *` parameters are changed to `_unichar *`. For example, this is the standard (non-Unicode) synopsis for the ASCII function `strrchr()`:

```
char * strrchr(const char *s, int c);
```

In the Unicode version, the synopsis changes to the following:

```
_unichar * _ustrrchr(const _unichar *s, int c);
```

The Unicode I/O functions treat strings and files as sequences of Unicode characters, instead of sequences of ASCII characters.

printf and scanf Format Conversions

Format strings for variations of Unicode functions `_uprintf()` and `_uscanf()` must be preceded by `L` to inform the compiler that the format string is a Unicode string. For example, use the format string `L"%s"` for a Unicode function instead of `"%s"`. A Unicode `_uprintf()` statement is of the following form:

```
_uprintf(L"prints a %s string \n",
         _unichar *unicode_string);
```

Printing ASCII Characters with Unicode Functions

The `%s` and `%c` conversions in `_uprintf()` and `_uscanf()` expect Unicode strings and characters as arguments. Use `%hc` and `%hs` instead of `%c` and `%s` to specify that the conversion is performed on an ASCII character. For example, to print a single ASCII character with `_uprintf()`, use the following:

```
_uprintf(L"Conversion performed on a single ASCII "
         "character %hc \n", ascichar);
```

Printing Unicode Characters with ASCII Functions

ANSI functions `printf()` and `scanf()` are enhanced to allow `%ls` and `%lc` to specify that the conversion is performed on a Unicode string or character. For example, to print a Unicode character with `printf()`, use the following:

```
printf("Conversion performed on a single Unicode "
      "character %lc \n", unichar);
```

In a few cases the parameters and return type of a Unicode function are identical to those of the equivalent ASCII function. These functions typically operate on a Unicode character instead of an ASCII character. Examples of such functions are `_ufgetc()` and `_ufputc()`, which respectively read and write a Unicode character to or from a file.

Dual-Mode Macros

For each Unicode function that has a corresponding ASCII function, there is also a dual-mode macro that you can use to call either function. For instance, `Uatoi` is the dual-mode macro for `uatoi()`. If `__UNICODE__` is defined, the macro calls the Unicode function; otherwise, it calls the ASCII function.

If you want the dual-mode macros to refer to the Unicode functions, define the `__UNICODE__` macro before including any header files. For example:

```
#define __UNICODE__
#include <stdio.h>
#include <string.h>
```

would make calls to dual-mode macros operate on Unicode characters for functions declared in `stdio.h` and `string.h`. Recompile the code without `#define __UNICODE__` to make the calls to the dual-mode macros operate on ASCII characters. The tables in the following sections show the dual-mode macro for each Unicode function.

Unicode Function in assert.h

Header file `assert.h` defines the macro `_uassert`, which puts diagnostics into a program. If a user-defined macro `NDEBUG` is defined at the point of inclusion of the header file `assert.h`, then `_uassert` has no effect.

Macro `_uassert` is identical to macro `assert`, except that it writes a Unicode string to standard error.

Unicode Functions in ctype.h

For more information about the contents of this header file, see [ctype.h](#) on page 29. For information about the ASCII functions, see [Function Reference](#) on page 49.

Unicode Function	ASCII Equivalent	Dual-Mode Macro	Description
<code>_uisalnum()</code>	isalnum()	<code>Uisalnum</code>	Test for alphanumeric Unicode character
<code>_uisalpha()</code>	isalpha()	<code>Uisalpha</code>	Test for alphabetic Unicode character
<code>_uisascii()</code>	isascii() and _isascii()	<code>Uisascii</code>	Test for ASCII character
<code>_uiscntrl()</code>	iscntrl()	<code>Uiscntrl</code>	Test for control Unicode character
<code>_uisdigit()</code>	isdigit()	<code>Uisdigit</code>	Test for numeric Unicode character
<code>_uisgraph()</code>	isgraph()	<code>Uisgraph</code>	Test for visible Unicode character
<code>_uislower()</code>	islower()	<code>Uislower</code>	Test for lowercase alphabetic Unicode character
<code>_uisodigit()</code>	isodigit()	<code>Uisodigit</code>	Test for octal numeric Unicode character
<code>_uisprint()</code>	isprint()	<code>Uispaint</code>	Test for printable Unicode character
<code>_uispunct()</code>	ispunct()	<code>Uispunct</code>	Test for punctuation Unicode character

Unicode Function	ASCII Equivalent	Dual-Mode Macro	Description
<code>_uiisspace()</code>	isspace()	<code>Uiisspace</code>	Test for whitespace Unicode character
<code>_uisupper()</code>	isupper()	<code>Uisupper</code>	Test for uppercase alphabetic Unicode character
<code>_uisxdigit()</code>	isxdigit()	<code>Uisxdigit</code>	Test for hexadecimal numeric Unicode character
<code>_tolower()</code>	tolower()	<code>Utolower</code>	Convert to lowercase Unicode character
<code>_toupper()</code>	toupper()	<code>Utoupper</code>	Convert to uppercase Unicode character

Unicode Functions in stdio.h

For more information about the contents of this header file, see [stdio.h](#) on page 40. For information about the ASCII functions, see [Function Reference](#) on page 49.

Unicode Function	ASCII Equivalent	Dual-Mode Macro	Description
<code>_u_bprintf()</code>	bprintf()	<code>U_bprintf</code>	Print to a Unicode string
<code>_ufdopen()</code>	fdopen() and _fdopen()	<code>Ufdopen</code>	Associate a stream with an open handle
<code>_ufgetc()</code>	fgetc()	<code>Ufgetc</code>	Get a Unicode character from a file
<code>_ufgetchar()</code>	fgetchar() and _fgetchar()	<code>Ufgetchar</code>	Get a Unicode character from <code>stdin</code>
<code>_ufgets()</code>	fgets()	<code>Ufgets</code>	Read a line of Unicode text from a file
<code>_ufopen()</code>	fopen()	<code>Ufopen</code>	Open a file
<code>_ufprintf()</code>	fprintf()	<code>Ufprintf</code>	Print Unicode characters to a file
<code>_ufputc()</code>	fputc()	<code>Ufputc</code>	Write a Unicode character to a file
<code>_ufputchar()</code>	fputchar() and _fputchar()	<code>Ufputchar</code>	Write a Unicode character to <code>stdout</code>
<code>_ufputs()</code>	fputs()	<code>Ufputs</code>	Write a Unicode string to a file
<code>_ufreopen()</code>	freopen()	<code>Ufreopen</code>	Open a file using a particular <code>FILE</code> variable
<code>_ufscanf()</code>	fscanf()	<code>Ufscanf</code>	Read Unicode values from a file
<code>_ugetc()</code>	getc()	<code>Ugetc</code>	Get a Unicode character from a file
<code>_ugetchar()</code>	getchar()	<code>Ugetchar</code>	Get a Unicode character from standard input
<code>_ugets()</code>	gets()	<code>Ugets</code>	Get a line of Unicode text from standard input
<code>_uperror()</code>	perror()	<code>Uperror</code>	Print error message
<code>_uprintf()</code>	printf()	<code>Uprintf</code>	Print to standard output
<code>_uputc()</code>	putc()	<code>Uputc</code>	Write a Unicode character to a file
<code>_uputchar()</code>	putchar()	<code>Uputchar</code>	Write a Unicode character to standard output
<code>_uputs()</code>	puts()	<code>Uputs</code>	Write a Unicode string to standard output
<code>_uscanf()</code>	scanf()	<code>Uscanf</code>	Read Unicode values from standard input
<code>_usprintf()</code>	sprintf()	<code>Usprintf</code>	Print to a Unicode string
<code>_usscanf()</code>	sscanf()	<code>Usscanf</code>	Read values from a Unicode string

Unicode Function	ASCII Equivalent	Dual-Mode Macro	Description
<code>_utmpnam()</code>	tmpnam()	<code>Utmpnam</code>	Generate a temporary file name
<code>_ungetc()</code>	ungetc()	<code>Uungetc</code>	Push a Unicode character back into an input stream
<code>_u_vbprintf()</code>	vbprintf()	<code>U_vbprintf</code>	Print to a Unicode string using variable arguments
<code>_uvfprintf()</code>	vfprintf()	<code>Uvfprintf</code>	Print to a file using Unicode variable arguments
<code>_uvfscanf()</code>	vfscanf() and vfscanf()	<code>Uvfscanf</code>	Read from a file using Unicode variable arguments
<code>_uvprintf()</code>	vprintf()	<code>Uvprintf</code>	Print to standard output using Unicode variable arguments
<code>_uvscanf()</code>	vscanf() and vscanf()	<code>Uvscanf</code>	Read from standard input using Unicode variable arguments
<code>_vsprintf()</code>	vsprintf()	<code>Uvsprintf</code>	Print to a string using Unicode variable arguments
<code>_vsscanf()</code>	vsscanf() and vsscanf()	<code>Uvsscanf</code>	Read from a string using Unicode variable arguments

Unicode Functions in stdlib.h

For more information about the contents of this header file, see [stdlib.h](#) on page 42. For information about the ASCII functions, see [Function Reference](#) on page 49.

Unicode Function	ASCII Equivalent	Dual-Mode Macro	Description
<code>_atouni()</code>	<i>None</i>	<i>None</i>	Convert ASCII string to a Unicode string
<code>_uatof()</code>	atof()	<code>Uatof</code>	Convert Unicode string to float
<code>_uatoi()</code>	atoi()	<code>Uatoi</code>	Convert Unicode string to int
<code>_uatol()</code>	atol()	<code>Uatol</code>	Convert Unicode string to long int
<code>_uecvt()</code>	ecvt() and _ecvt()	<code>Uecvt</code>	Convert float to Unicode character string
<code>_ufcvt()</code>	fcvt() and _fcvt()	<code>Ufcvt</code>	Convert float to Unicode character string
<code>_ugcvt()</code>	gcvt() and _gcvt()	<code>Ugcvt</code>	Convert float to Unicode character string
<code>_itoa()</code>	itoa() and _itoa()	<code>Uitoa</code>	Convert int to Unicode string
<code>_u_ltoa()</code>	ltoa() and _ltoa()	<code>Ultoa</code>	Convert long int to Unicode string
<code>_ustrtod()</code>	strtod()	<code>Ustrtod</code>	Convert Unicode string to double
<code>_ustrtol()</code>	strtol()	<code>Ustrtol</code>	Convert Unicode string to long
<code>_ustrtoul()</code>	strtoul()	<code>Ustrtoul</code>	Convert Unicode string to unsigned long
<code>_ultoa()</code>	ultoa() and _ultoa()	<code>Uultoa</code>	Convert unsigned long int to Unicode string
<code>_utooa()</code>	utoa() and _utoa()	<code>Uutoa</code>	Convert unsigned int to Unicode string

Unicode Functions in string.h

For more information about the contents of this header file, see [string.h](#) on page 44. For information about the ASCII functions, see [Function Reference](#) on page 49.

Unicode Function	ASCII Equivalent	Dual-Mode Macro	Description
<code>_uchrcpy()</code>	memccpy() and _memccpy()	<code>Uchrcpy</code>	Copy memory until count or character
<code>_uchrchr()</code>	memchr()	<code>Uchrchr</code>	Find Unicode character in an area of memory
<code>_uchrcmp()</code>	memcmp()	<code>Uchrcmp</code>	Compare two areas of memory
<code>_uchrcpy()</code>	memcpy()	<code>Uchrcpy</code>	Copy from one place in memory to another
<code>_uchricmp()</code>	memicmp() and _memicmp()	<code>Uchricmp</code>	Compare memory, ignoring case
<code>_uchrset()</code>	memset()	<code>Uchrset</code>	Duplicate one Unicode character across an area of memory
<code>_ustrcat()</code>	strcat()	<code>Ustreat</code>	Concatenate two Unicode strings
<code>_ustrchr()</code>	strchr()	<code>Ustrchr</code>	Find first occurrence of a Unicode character in a string
<code>_ustrcmp()</code>	strcmp()	<code>Ustrcmp</code>	Compare one Unicode string to another
<code>_ustrcmpi()</code>	strcasecmp() and _strcasecmp()	<code>Ustrcmpi</code>	Compare Unicode strings, ignoring case
<code>_ustrcpy()</code>	strcpy()	<code>Ustrcpy</code>	Copy Unicode characters of one string into another
<code>_ustrespn()</code>	strrespn()	<code>Ustrespn</code>	Length of the prefix of a Unicode string that has no characters that match another string
<code>_ustrdup()</code>	strdup() and _strdup()	<code>Ustrdup</code>	Duplicate a Unicode string
<code>_ustrrror()</code>	strerror() and _strerror()	<code>Ustrrror</code>	Map error code to Unicode error-message string
<code>_ustricmp()</code>	strcmpi() and _stricmp()	<code>Ustricmp</code>	Compare Unicode strings, ignoring case
<code>_ustrlen()</code>	strlen()	<code>Ustrlen</code>	Length of a Unicode string
<code>_ustrlwr()</code>	strlwr() and _strlwr()	<code>Ustrlwr</code>	Convert a Unicode string to lowercase
<code>_ustrncat()</code>	strncat() and _strncat()	<code>Ustrncat</code>	Append characters of one Unicode string onto another with a limit
<code>_ustrncmp()</code>	strncmp()	<code>Ustrncmp</code>	Compare Unicode characters of one string to another
<code>_ustrncpy()</code>	strncpy()	<code>Ustrncpy</code>	Copy characters from one Unicode string into another
<code>_ustrnicmp()</code>	strnicmp() and _strnicmp()	<code>Ustrnicmp</code>	Compare parts of Unicode strings, ignoring case
<code>_ustrnset()</code>	strnset() and _strnset()	<code>Ustrnset</code>	Set part of a Unicode string to a character
<code>_ustrpbrk()</code>	strpbrk()	<code>Ustrpbrk</code>	Find first occurrence of any Unicode character of one string in another
<code>_ustrrchr()</code>	strrchr()	<code>Ustrrchr</code>	Find last occurrence of a Unicode character in a string
<code>_ustrrev()</code>	strrev() and _strrev()	<code>Ustrrev</code>	Reverse the Unicode characters in a string

Unicode Function	ASCII Equivalent	Dual-Mode Macro	Description
<code>_ustrset()</code>	strset() and _strset()	<code>Ustrset</code>	Set a Unicode string to a character
<code>_ustrspn()</code>	strspn()	<code>Ustrspn</code>	Length of the prefix of a Unicode string composed of characters that match those from another string
<code>_ustrstr()</code>	strstr()	<code>Ustrstr</code>	Find first occurrence of one Unicode string within another
<code>_ustrtok()</code>	strtok()	<code>Ustrtok</code>	Divide Unicode string into tokens
<code>_strupr()</code>	strupr() and _strupr()	<code>Ustrupr</code>	Convert a Unicode string to uppercase

Unicode Functions in time.h

For more information about the contents of this header file, see [time.h](#) on page 46. For information about the ASCII functions, see [Function Reference](#) on page 49.

Unicode Function	ASCII Equivalent	Dual-Mode Macro	Description
<code>_uasctime()</code>	asctime() and _asctime()	<code>Uasctime</code>	Convert a struct tm to printable form
<code>_u_asctime()</code>	asctime() and _asctime()	<code>U_asctime</code>	Convert a struct tm to printable form
<code>_uctime()</code>	ctime() and _ctime()	<code>Uctime</code>	Convert a time_t to printable form
<code>_u_ctime()</code>	ctime() and _ctime()	<code>U_ctime</code>	Convert a time_t to printable form

Appendix C — ANSI Compliance

In This Appendix

- [ANSI-Compliant Functions](#)
- [ANSI C99-Compliant Functions](#)
- [Extra-ANSI Functions](#)
- [Non-ANSI Functions](#)

NOTE For definitions of the terms *ANSI compliant*, *Extra-ANSI*, and *Non-ANSI*, see [Names and Terms](#) on page 17.

ANSI-Compliant Functions

- [abort\(\) — Terminate a program abnormally](#)
- [abs\(\) — Calculate the absolute value of an integer](#)
- [acos\(\) — Arc cosine](#)
- [asctime\(\) and _asctime\(\) — Convert a struct tm to printable form](#)
- [asin\(\) — Arc sine](#)
- [atan\(\) — Arc tangent](#)
- [atan2\(\) — Arc tangent of the angle defined by a point](#)
- [atexit\(\) — Register a function for execution at program termination](#)
- [atof\(\) — Convert a prefix of a string to floating point](#)
- [atoi\(\) — Convert a prefix of a string to an int](#)
- [atol\(\) — Convert a prefix of a string to a long int](#)
- [bsearch\(\) — Perform a binary search](#)
- [calloc\(\) — Dynamically allocate zero-initialized storage for objects](#)
- [ceil\(\) — Calculate smallest integer greater than or equal to a value](#)
- [clearerr\(\) — Clear end-of-file and error flags](#)
- [clock\(\) — Report elapsed processor time](#)
- [cos\(\) — Cosine](#)
- [coshf\(\) — Hyperbolic cosine using single-precision floating point](#)
- [ctime\(\) and _ctime\(\) — Convert a time_t value to printable form](#)
- [difftime\(\) — Calculate difference between two times](#)
- [div\(\) — Perform integer division with remainder](#)
- [exit\(\) — Terminate a program normally](#)
- [exp\(\) — Calculate exponential \(e to the power x\)](#)
- [fabs\(\) — Calculate absolute value of a floating-point number using double precision](#)
- [fclose\(\) — Close a file](#)
- [feof\(\) — Test for end-of-file](#)
- [ferror\(\) — Test for a read or write error on a file](#)
- [fflush\(\) — Flush a file's buffer](#)
- [fgetc\(\) — Read \(get\) a character from a file](#)
- [fgetpos\(\) — Determine file position](#)
- [fgets\(\) — Read a line of text \(string\) from a file](#)
- [floor\(\) — Calculate largest integer not greater than a value](#)
- [fmod\(\) — Return floating-point remainder](#)

- [fopen\(\) — Open a file](#)
- [fprintf\(\) — Print to a file](#)
- [fputc\(\) — Write a character to a file](#)
- [fputs\(\) — Write a string to a file](#)
- [fread\(\) — Read from a file](#)
- [free\(\) — Release storage allocated by calloc\(\), malloc\(\), or realloc\(\)](#)
- [freopen\(\) — Open a file using an existing FILE variable](#)
- [frexp\(\) — Break a double into fraction and exponent](#)
- [fscanf\(\) — Read values from a file](#)
- [fseek\(\) — Move file pointer to new location in file](#)
- [fsetpos\(\) — Set file position](#)
- [ftell\(\) — Determine current value of a file pointer](#)
- [fwrite\(\) — Write to a file](#)
- [getc\(\) — Get a character from a file](#)
- [getchar\(\) — Get a character from standard input](#)
- [getenv\(\) — Get the value of an environment variable](#)
- [gets\(\) — Read a line of text \(string\) from standard input](#)
- [gmtime\(\) and gmtime\(\) — Convert a time_t value to a struct tm, adjusted to UTC](#)
- [isalnum\(\) — Test for alphanumeric character](#)
- [isalpha\(\) — Test for alphabetic character](#)
- [iscntrl\(\) — Test for control character](#)
- [isdigit\(\) — Test for numeric character](#)
- [isgraph\(\) — Test for visible character](#)
- [islower\(\) — Test for lowercase alphabetic character](#)
- [isprint\(\) — Test for printable character](#)
- [ispunct\(\) — Test for punctuation character](#)
- [isspace\(\) — Test for whitespace character](#)
- [isupper\(\) — Test for uppercase alphabetic character](#)
- [isxdigit\(\) — Test for hexadecimal numeric character](#)
- [labs\(\) — Calculate the absolute value of a long int](#)
- [ldexp\(\) — Multiply by a power of 2](#)
- [ldiv\(\) — Perform long int division with remainder](#)
- [localeconv\(\) — Query current locale for numeric format](#)
- [localtime\(\) and localtime\(\) — Convert a time_t value to a struct tm](#)

- [log\(\)](#) — [Calculate natural logarithm \(base e\)](#)
- [log10\(\)](#) — [Calculate common logarithm \(base 10\)](#)
- [longjmp\(\)](#) — [Execute a non-local jump](#)
- [malloc\(\)](#) — [Dynamically allocate uninitialized storage](#)
- [mblen\(\)](#) — [Determine length of a multibyte character](#)
- [mbstowcs\(\)](#) — [Convert a multibyte string to wchar_t array](#)
- [mbtowc\(\)](#) — [Convert a multibyte character to wchar_t code](#)
- [memchr\(\)](#) — [Find a character in an area of memory](#)
- [memcmp\(\)](#) — [Compare two areas of memory](#)
- [memcpy\(\)](#) — [Copy from one place in memory to another](#)
- [memmove\(\)](#) — [Copy memory nondestructively](#)
- [memset\(\)](#) — [Duplicate a character across an area of memory](#)
- [mktime\(\)](#) — [Convert a struct tm to a time_t value](#)
- [modf\(\)](#) — [Break a double into integer and fractional parts](#)
- [offsetof\(\)](#) — [Determine offset of a structure member](#)
- [perror\(\)](#) — [Print an error message](#)
- [pow\(\)](#) — [Raise a double to a power](#)
- [printf\(\)](#) — [Print to stdout](#)
- [putc\(\)](#) — [Write a character to a file](#)
- [putchar\(\)](#) — [Write a character to standard output](#)
- [puts\(\)](#) — [Write a string to standard output](#)
- [qsort\(\)](#) — [Perform a quicksort](#)
- [raise\(\)](#) — [Raise a signal](#)
- [rand\(\)](#) — [Generate a pseudo-random number](#)
- [realloc\(\)](#) — [Reallocate storage allocated by calloc\(\) or malloc\(\)](#)
- [remove\(\)](#) — [Delete a file](#)
- [rename\(\)](#) — [Change the name of a file](#)
- [rewind\(\)](#) — [Seek to the beginning of a file](#)
- [scanf\(\)](#) — [Read values from stdin](#)
- [setbuf\(\)](#) — [Specify a buffer for a stream](#)
- [setjmp\(\)](#) — [Save a reference to the current calling environment for a subsequent non-local jump](#)
- [setlocale\(\)](#) — [Set the current locale](#)
- [setvbuf\(\)](#) — [Control file buffering](#)

- [signal\(\) — Set up a signal handler](#)
- [sin\(\) — Sine](#)
- [sinh\(\) — Hyperbolic sine](#)
- [sprintf\(\) — Print to a string](#)
- [sqrt\(\) — Square root](#)
- [rand\(\) — Seed the pseudo-random number generator](#)
- [scanf\(\) — Read values from a string](#)
- [strcat\(\) — Concatenate two strings](#)
- [strchr\(\) — Find the first occurrence of a character in a string](#)
- [strcmp\(\) — Compare one string to another](#)
- [strcoll\(\) — Compare strings based on current locale](#)
- [strcpy\(\) — Copy the characters of one string into another](#)
- [strcspn\(\) — Determine the length of the prefix of a string not containing any characters from another string](#)
- [strerror\(\) and _strerror\(\) — Map error code; append error-message string to user message](#)
- [strftime\(\) — Format time and date string](#)
- [strlen\(\) — Determine the length of a string](#)
- [strncat\(\) and _strncat\(\) — Append characters of one string to another, up to some limit n](#)
- [strcmp\(\) — Compare characters of one string to characters of another](#)
- [strncpy\(\) — Copy a number of characters from one string into another](#)
- [strpbrk\(\) — Find the first occurrence of any character of one string in another](#)
- [strrchr\(\) — Find the last occurrence of a character in a string](#)
- [strspn\(\) — Determine the length of the prefix of a string composed entirely of characters from another string](#)
- [strstr\(\) — Find the first occurrence of one string within another](#)
- [strtod\(\) — Convert a string to a double](#)
- [strtok\(\) — Divide a string into tokens](#)
- [strtol\(\) — Convert a string to a long](#)
- [strtoul\(\) — Convert a string to an unsigned long](#)
- [strxfrm\(\) — Transform string to locale-independent form](#)
- [system\(\) — Pass a command to the operating system](#)
- [tan\(\) — Tangent](#)
- [tanh\(\) — Hyperbolic tangent](#)
- [time\(\) — Get the current time and date](#)
- [tmpfile\(\) and _tmpfile\(\) — Create a temporary file](#)

- [`tmpnam\(\)`](#) — [Generate a string to be used as a temporary file name](#)
- [`tolower\(\)`](#) — [Convert to lowercase](#)
- [`toupper\(\)`](#) — [Convert to uppercase](#)
- [`ungetc\(\)`](#) — [Push a character back into an input stream](#)
- [`va_arg\(\)`](#) — [Get the next argument in a variable series of arguments](#)
- [`va_end\(\)`](#) — [Terminate va_list\(\) processing](#)
- [`va_start\(\)`](#) — [Initialize an object of type va_list](#)
- [`vfprintf\(\)`](#) — [Print to a file using var-arg macros](#)
- [`vprintf\(\)`](#) — [Print to `stdout` using var-arg macros](#)
- [`vsprintf\(\)`](#) — [Print to a string using var-arg macros](#)
- [`wstombs\(\)`](#) — [Convert a `wchar_t` array to a multibyte string](#)
- [`wctomb\(\)`](#) — [Convert a `wchar_t` code to a multibyte character](#)

ANSI C99-Compliant Functions

- [`acosf\(\)`](#) — [Arc cosine using single-precision floating point](#)
- [`asinf\(\)`](#) — [Arc sine using single-precision floating point](#)
- [`atan2f\(\)`](#) — [Arc tangent of the angle defined by a point, using single-precision floating point](#)
- [`atanf\(\)`](#) — [Arc tangent using single-precision floating point](#)
- [`ceilf\(\)`](#) — [Calculate smallest integer greater than or equal to a value using single-precision floating point](#)
- [`cosf\(\)`](#) — [Cosine using single-precision floating point](#)
- [`coshf\(\)`](#) — [Hyperbolic cosine using single-precision floating point](#)
- [`expf\(\)`](#) — [Calculate exponential \(\$e\$ to the power \$x\$ \) using single-precision floating point](#)
- [`fabsf\(\)`](#) — [Calculate absolute value of a floating-point number using single precision](#)
- [`floorf\(\)`](#) — [Calculate largest integer not greater than a value using single-precision floating point](#)
- [`fmodf\(\)`](#) — [Return floating-point remainder using single precision](#)
- [`frexpf\(\)`](#) — [Break a float into fraction and exponent](#)
- [`ldexpf\(\)`](#) — [Multiply by a power of 2 using single-precision floating point](#)
- [`log10f\(\)`](#) — [Calculate common logarithm \(base 10\) using single-precision floating point](#)
- [`logf\(\)`](#) — [Calculate natural logarithm \(base \$e\$ \) using single-precision floating point](#)
- [`modff\(\)`](#) — [Break a float into integer and fractional parts](#)
- [`powf\(\)`](#) — [Raise a float to a power](#)
- [`sinf\(\)`](#) — [Sine using single-precision floating point](#)
- [`sinhf\(\)`](#) — [Hyperbolic sine using single-precision floating point](#)

- [sqrtf\(\)](#) — [Square root using single-precision floating point](#)
- [tanf\(\)](#) — [Tangent using single-precision floating point](#)
- [tanhf\(\)](#) — [Hyperbolic tangent using single-precision floating point](#)

Extra-ANSI Functions

- [abs\(\)](#) — [Calculate the absolute value of any arithmetic type](#)
- [access\(\)](#) and [_access\(\)](#) — [Test access rights of a file](#)
- [acosh\(\)](#) and [_acosh\(\)](#) — [Hyperbolic arc cosine](#)
- [alloca\(\)](#) and [_alloca\(\)](#) — [Allocate memory on the stack](#)
- [asctime\(\)](#) and [_asctime\(\)](#) — [Convert a struct tm to printable form](#)
- [asinh\(\)](#) and [_asinh\(\)](#) — [Hyperbolic arc sine](#)
- [atanh\(\)](#) and [_atanh\(\)](#) — [Hyperbolic arc tangent](#)
- [atold\(\)](#) — [Convert a prefix of a string to a long double](#)
- [bprintf\(\)](#) — [Print to a string of given size](#)
- [cabs\(\)](#) and [_cabs\(\)](#) — [Compute complex absolute value](#)
- [cgets\(\)](#) and [_cgets\(\)](#) — [Read a line of text \(string\) from the keyboard](#)
- [chdir\(\)](#) and [_chdir\(\)](#) — [Change the current working directory](#)
- [chmod\(\)](#) and [_chmod\(\)](#) — [Alter the permission setting for a file](#)
- [chsize\(\)](#) and [_chsize\(\)](#) — [Extend or truncate the length of a file](#)
- [close\(\)](#) and [_close\(\)](#) — [Close a file handle](#)
- [closedir\(\)](#) — [Close a POSIX directory stream](#)
- [cprintf\(\)](#) and [_cprintf\(\)](#) — [Print to screen](#)
- [cputs\(\)](#) and [_cputs\(\)](#) — [Write a string to the screen](#)
- [creat\(\)](#) and [_creat\(\)](#) — [Create an empty file](#)
- [crotl\(\)](#) and [_crotr\(\)](#), [_crotl\(\)](#) and [_crotr\(\)](#) — [Rotate a char left or right](#)
- [escanf\(\)](#) and [_escanf\(\)](#) — [Read from standard input](#)
- [ctime\(\)](#) and [_ctime\(\)](#) — [Convert a time_t value to printable form](#)
- [dieetombsbin\(\)](#) and [_dieetombsbin\(\)](#) — [Convert IEEE double to Microsoft double](#)
- [disable\(\)](#) — [Disable interrupts](#)
- [div0\(\)](#) — [Replaceable handler for integer division by zero \(ARC only\)](#)
- [dmsbintoieee\(\)](#) and [_dmsbintoieee\(\)](#) — [Convert Microsoft double to IEEE double](#)
- [dup\(\)](#) and [_dup\(\)](#) — [Duplicate a file handle](#)
- [dup2\(\)](#) and [_dup2\(\)](#) — [Duplicate one file handle into another](#)
- [ecvt\(\)](#) and [_ecvt\(\)](#) — [Convert a floating-point number to a character string](#)

- [_enable\(\)](#) — [Enable interrupts](#)
- [eof\(\) and _eof\(\)](#) — [Test a file handle for end-of-file status](#)
- [_exit\(\)](#) — [Terminate a program immediately](#)
- [fcloseall\(\) and _fcloseall\(\)](#) — [Close all files](#)
- [fcvt\(\) and _fcvt\(\)](#) — [Convert a floating-point number to a character string](#)
- [fdopen\(\) and _fdopen\(\)](#) — [Associate a stream with an open handle](#)
- [fgetchar\(\) and _fgetchar\(\)](#) — [Get a character from standard input](#)
- [fieetombsin\(\) and _fieetombsin\(\)](#) — [Convert IEEE float to Microsoft format](#)
- [filelength\(\) and _filelength\(\)](#) — [Determine length of an open file](#)
- [fileno\(\) and _fileno\(\)](#) — [Get file handle currently associated with a stream](#)
- [findclose\(\) and _findfirst\(\) and _findnext\(\)](#) — [Find a file whose name matches a specification](#)
- [findclose\(\) and _findfirst\(\) and _findnext\(\), _findclose\(\) and _findfirst\(\) and _findnext\(\), _findclose\(\) and _findfirst\(\) and _findnext\(\)](#) — [Find a file whose name matches a specification](#)
- [flushall\(\) and _flushall\(\)](#) — [Clear input buffers and flush output buffers](#)
- [fmsbintoieee\(\) and _fmsbintoieee\(\)](#) — [Convert Microsoft float to IEEE float](#)
- [FP_OFF\(\), FP_SEG\(\) and _FP_OFFSET\(\), _FP_SEG\(\)](#) — [Get offset and segment of a pointer](#)
- [fputchar\(\) and _fputchar\(\)](#) — [Write a character to standard output](#)
- [ftime\(\) and _ftime\(\)](#) — [Get current time values](#)
- [fullpath\(\)](#) — [Convert relative pathname to absolute pathname](#)
- [gcvt\(\) and _gcvt\(\)](#) — [Convert a floating-point number to a character string](#)
- [getcwd\(\) and _getcwd\(\)](#) — [Get full pathname of the current working directory](#)
- [getpid\(\) and _getpid\(\)](#) — [Get unique ID of the calling process](#)
- [getw\(\) and _getw\(\)](#) — [Get an integer from a file](#)
- [gmtime\(\) and _gmtime\(\)](#) — [Convert a time_t value to a struct tm, adjusted to UTC](#)
- [heapchk\(\)](#) — [Check heap for consistency](#)
- [heapset\(\)](#) — [Fill free memory locations in the heap](#)
- [heapwalk\(\)](#) — [Walk through the heap](#)
- [hypot\(\) and _hypot\(\)](#) — [Hypotenuse of a triangle](#)
- [inline\(\)](#) — [Place bytes of code in line](#)
- [invalidate_dcache\(\)](#) — [Invalidate data cache \(ARC only\)](#)
- [invalidate_icache\(\)](#) — [Invalidate instruction cache \(ARC only\)](#)
- [isascii\(\) and _isascii\(\)](#) — [Test for ASCII character](#)
- [isatty\(\) and _isatty\(\)](#) — [Identify character devices](#)
- [_isodigit\(\)](#) — [Test for octal numeric character](#)

- [itoa\(\) and _itoa\(\)](#) — [Convert an int value to an ASCII string](#)
- [j00, j10, jn\(\) and _j00, _j10, _jn\(\)](#) — [Bessel function of the first kind](#)
- [ldecvt\(\)](#) — [Convert a long double to a character string](#)
- [ldfcvt\(\)](#) — [Convert a long double to a character string](#)
- [lfind\(\) and _lfind\(\)](#) — [Search a linear list for a matching value, without modifying the list](#)
- [localtime\(\) and _localtime\(\)](#) — [Convert a time_t value to a struct tm](#)
- [log2\(\) and _log2\(\)](#) — [Calculate logarithm \(base 2\)](#)
- [lrotl\(\) and _lrotr\(\), lrotl\(\) and _lrotr\(\)](#) — [Rotate a long int left or right](#)
- [lsearch\(\) and _lsearch\(\)](#) — [Search a linear list for a matching value, modifying if necessary](#)
- [lseek\(\) and _lseek\(\)](#) — [Change the position in a file](#)
- [ltoa\(\) and _ltoa\(\)](#) — [Convert a long int value to an ASCII string](#)
- [makepath\(\)](#) — [Create a path string from components](#)
- [matherr\(\), matherrx\(\) and _matherr\(\), _matherrx\(\)](#) — [Provide standard error reporting for math functions](#)
- [max\(\)](#) — [Determine maximum of any arithmetic type](#)
- [memccpy\(\) and _memccpy\(\)](#) — [Copy memory until count or character](#)
- [memcmp\(\) and _memcmp\(\)](#) — [Compare bytes ignoring case](#)
- [min\(\)](#) — [Determine minimum of any arithmetic type](#)
- [mkdir\(\) and _mkdir\(\)](#) — [Create a new directory](#)
- [mktemp\(\) and _mktemp\(\)](#) — [Create a unique file name \(does not create or open files\)](#)
- [msize\(\)](#) — [Get the size of a memory block in the heap](#)
- [open\(\) and _open\(\)](#) — [Open a file handle with a pathname](#)
- [opendir\(\)](#) — [Open a POSIX directory stream](#)
- [putch\(\) and _putch\(\)](#) — [Write a character to the screen](#)
- [putenv\(\) and _putenv\(\)](#) — [Add or modify an environment variable](#)
- [putw\(\) and _putw\(\)](#) — [Write an integer to a file](#)
- [read\(\) and _read\(\)](#) — [Read data from a file using a handle](#)
- [readdir\(\)](#) — [Read a POSIX directory stream](#)
- [rewinddir\(\)](#) — [Reset a POSIX directory stream](#)
- [rmdir\(\) and _rmdir\(\)](#) — [Remove a directory](#)
- [rmemcpy\(\)](#) — [Copy from one place in memory to another](#)
- [rotl\(\) and _rotr\(\), rotl\(\) and _rotr\(\)](#) — [Rotate an int left or right](#)
- [strcpy\(\)](#) — [Copy one string onto another](#)
- [rstrncpy\(\)](#) — [Copy a number of characters from one string into another](#)

- [searchenv\(\)](#) — [Search for a file using an environment variable](#)
- [searchstr\(\)](#) — [Search for a file in a list of directories](#)
- [set_matherr\(\)](#) — [User defined math error function](#)
- [setmode\(\)](#) — [Set stream mode \(`fmode` \) to text or binary](#)
- [setvect10\(\)](#) — [Set low-priority interrupt to jump to target \(ARC only\)](#)
- [setvect20\(\)](#) — [Set mid- or high-priority interrupt to jump to target \(ARC only\)](#)
- [sleep\(\)](#) — [Temporarily suspend program execution](#)
- [spawn*\(\)](#) and [spawn*\(\)](#) — [Execute a child process](#)
- [splitpath\(\)](#) — [Split a pathname into its components](#)
- [srotl\(\)](#) and [srotr\(\)](#), [srotl\(\)](#) and [srotr\(\)](#) — [Rotate a short int left or right](#)
- [stat\(\)](#) and [stat\(\)](#) — [Get information about a file or directory](#)
- [strcats\(\)](#) — [Concatenate multiple strings up to a limited number of characters](#)
- [strcmpi\(\)](#) and [strcmpi\(\)](#) — [Compare strings, ignoring case](#)
- [strdate\(\)](#) — [Convert the current date to a string](#)
- [strdup\(\)](#) and [strdup\(\)](#) — [Make a copy of a string in the heap](#)
- [strerror\(\)](#) and [strerror\(\)](#) — [Map error code; append error-message string to user message](#)
- [stricmp\(\)](#) and [stricmp\(\)](#) — [Compare strings, ignoring case](#)
- [strlwr\(\)](#) and [strlwr\(\)](#) — [Make string all lowercase](#)
- [strneat\(\)](#) and [strneat\(\)](#) — [Append characters of one string to another, up to some limit n](#)
- [strnicmp\(\)](#) and [strnicmp\(\)](#) — [Compare strings, ignoring case](#)
- [strnset\(\)](#) and [strnset\(\)](#) — [Fill part or all of a string with any character](#)
- [strrev\(\)](#) and [strrev\(\)](#) — [Reverse the order of characters in a string](#)
- [strset\(\)](#) and [strset\(\)](#) — [Fill a string with any character](#)
- [strftime\(\)](#) — [Convert current time of day to a string](#)
- [strupr\(\)](#) and [strupr\(\)](#) — [Make string all uppercase](#)
- [swab\(\)](#) and [swab\(\)](#) — [Swap bytes in a word](#)
- [tell\(\)](#) and [tell\(\)](#) — [Report the current position in a file](#)
- [tempnam\(\)](#) — [Generate a temporary file name in a directory](#)
- [tmpfile\(\)](#) and [tmpfile\(\)](#) — [Create a temporary file](#)
- [tolower\(\)](#) — [Convert to lowercase without testing for uppercase](#)
- [toupper\(\)](#) — [Convert to uppercase without testing for lowercase](#)
- [tzset\(\)](#) and [tzset\(\)](#) — [Set local time-zone values](#)
- [udiv0\(\)](#) — [Replaceable handler for unsigned integer division by zero \(ARC only\)](#)
- [ultoa\(\)](#) and [ultoa\(\)](#) — [Convert an unsigned long to an ASCII string](#)

- [umask\(\) and _umask\(\)](#) — [Set global file permission mask](#)
- [ungetch\(\) and _ungetch\(\)](#) — [Push a character back into the input buffer](#)
- [unlink\(\) and _unlink\(\)](#) — [Delete a file](#)
- [utime\(\) and _utime\(\)](#) — [Update a file's date/time stamp](#)
- [utoa\(\) and _utoa\(\)](#) — [Convert an unsigned int to an ASCII string](#)
- [vprintf\(\)](#) — [Print to a string of given size using var-arg macros](#)
- [vdmemcpy\(\)](#) — [Copy to a volatile destination \(ARC only\)](#)
- [vfscanf\(\) and vfscanf\(\)](#) — [Read values from a file using var-arg macros](#)
- [vmemset\(\)](#) — [Duplicate a character to a volatile destination \(ARC only\)](#)
- [vscanf\(\) and vscanf\(\)](#) — [Read values from `stdin` using var-arg macros](#)
- [vsdmemcpy\(\)](#) — [Copy from a volatile source to a volatile destination \(ARC only\)](#)
- [vsMemcpy\(\)](#) — [Copy from a volatile source \(ARC only\)](#)
- [vsscanf\(\) and vsscanf\(\)](#) — [Read values from a string using var-arg macros](#)
- [write\(\) and _write\(\)](#) — [Write data to a file, unbuffered](#)
- [y0\(\), y1\(\), yn\(\) and _y0\(\), _y1\(\), _yn\(\)](#) — [Bessel function of the second kind](#)

Non-ANSI Functions

- [access\(\) and _access\(\)](#) — [Test access rights of a file](#)
- [acosh\(\) and _acosh\(\)](#) — [Hyperbolic arc cosine](#)
- [alloca\(\) and _alloca\(\)](#) — [Allocate memory on the stack](#)
- [asinh\(\) and _asinh\(\)](#) — [Hyperbolic arc sine](#)
- [assert\(\)](#) — [Abort if an assertion is false](#)
- [atanh\(\) and _atanh\(\)](#) — [Hyperbolic arc tangent](#)
- [cabs\(\) and _cabs\(\)](#) — [Compute complex absolute value](#)
- [cgets\(\) and _cgets\(\)](#) — [Read a line of text \(string\) from the keyboard](#)
- [chdir\(\) and _chdir\(\)](#) — [Change the current working directory](#)
- [chmod\(\) and _chmod\(\)](#) — [Alter the permission setting for a file](#)
- [chsize\(\) and _chsize\(\)](#) — [Extend or truncate the length of a file](#)
- [close\(\) and _close\(\)](#) — [Close a file handle](#)
- [cprintf\(\) and _cprintf\(\)](#) — [Print to screen](#)
- [cputs\(\) and _cputs\(\)](#) — [Write a string to the screen](#)
- [creat\(\) and _creat\(\)](#) — [Create an empty file](#)
- [escanf\(\) and _escanf\(\)](#) — [Read from standard input](#)
- [dieetombsbin\(\) and _dieetombsbin\(\)](#) — [Convert IEEE double to Microsoft double](#)

- [dmsbintoieee\(\) and _dmsbintoieee\(\)](#) — [Convert Microsoft double to IEEE double](#)
- [dup\(\) and _dup\(\)](#) — [Duplicate a file handle](#)
- [dup2\(\) and _dup2\(\)](#) — [Duplicate one file handle into another](#)
- [ecvt\(\) and _ecvt\(\)](#) — [Convert a floating-point number to a character string](#)
- [eof\(\) and _eof\(\)](#) — [Test a file handle for end-of-file status](#)
- [fcloseall\(\) and _fcloseall\(\)](#) — [Close all files](#)
- [fcvt\(\) and _fcvt\(\)](#) — [Convert a floating-point number to a character string](#)
- [fdopen\(\) and _fdopen\(\)](#) — [Associate a stream with an open handle](#)
- [fgetchar\(\) and _fgetchar\(\)](#) — [Get a character from standard input](#)
- [fieetomsbin\(\) and _fieetomsbin\(\)](#) — [Convert IEEE float to Microsoft format](#)
- [filelength\(\) and _filelength\(\)](#) — [Determine length of an open file](#)
- [fileno\(\) and _fileno\(\)](#) — [Get file handle currently associated with a stream](#)
- [flushall\(\) and _flushall\(\)](#) — [Clear input buffers and flush output buffers](#)
- [fmsbintoieee\(\) and _fmsbintoieee\(\)](#) — [Convert Microsoft float to IEEE float](#)
- [FP_OFF\(\), FP_SEG\(\) and _FP_OFFSET\(\), _FP_SEG\(\)](#) — [Get offset and segment of a pointer](#)
- [fputchar\(\) and _fputchar\(\)](#) — [Write a character to standard output](#)
- [ftime\(\) and _ftime\(\)](#) — [Get current time values](#)
- [gcvt\(\) and _gcvt\(\)](#) — [Convert a floating-point number to a character string](#)
- [getcwd\(\) and _getcwd\(\)](#) — [Get full pathname of the current working directory](#)
- [getpid\(\) and _getpid\(\)](#) — [Get unique ID of the calling process](#)
- [getw\(\) and _getw\(\)](#) — [Get an integer from a file](#)
- [hypot\(\) and _hypot\(\)](#) — [Hypotenuse of a triangle](#)
- [isascii\(\) and _isascii\(\)](#) — [Test for ASCII character](#)
- [isatty\(\) and _isatty\(\)](#) — [Identify character devices](#)
- [itoa\(\) and _itoa\(\)](#) — [Convert an int value to an ASCII string](#)
- [j0\(\), j1\(\), jn\(\) and _j0\(\), _j1\(\), _jn\(\)](#) — [Bessel function of the first kind](#)
- [lfind\(\) and _lfind\(\)](#) — [Search a linear list for a matching value, without modifying the list](#)
- [log2\(\) and _log2\(\)](#) — [Calculate logarithm \(base 2\)](#)
- [lsearch\(\) and _lsearch\(\)](#) — [Search a linear list for a matching value, modifying if necessary](#)
- [lseek\(\) and _lseek\(\)](#) — [Change the position in a file](#)
- [ltoa\(\) and _ltoa\(\)](#) — [Convert a long int value to an ASCII string](#)
- [matherr\(\), matherrx\(\) and _matherr\(\), _matherrx\(\)](#) — [Provide standard error reporting for math functions](#)
- [memccpy\(\) and _memccpy\(\)](#) — [Copy memory until count or character](#)

- [memicmp\(\) and _memicmp\(\)](#) — [Compare bytes ignoring case](#)
- [mkdir\(\) and _mkdir\(\)](#) — [Create a new directory](#)
- [mktemp\(\) and _mktemp\(\)](#) — [Create a unique file name \(does not create or open files\)](#)
- [open\(\) and _open\(\)](#) — [Open a file handle with a pathname](#)
- [putch\(\) and _putch\(\)](#) — [Write a character to the screen](#)
- [putenv\(\) and _putenv\(\)](#) — [Add or modify an environment variable](#)
- [putw\(\) and _putw\(\)](#) — [Write an integer to a file](#)
- [read\(\) and _read\(\)](#) — [Read data from a file using a handle](#)
- [rmdir\(\) and _rmdir\(\)](#) — [Remove a directory](#)
- [spawn*\(\) and _spawn*\(\)](#) — [Execute a child process](#)
- [stat\(\) and _stat\(\)](#) — [Get information about a file or directory](#)
- [strcmpl\(\) and _strcmpl\(\)](#) — [Compare strings, ignoring case](#)
- [strdup\(\) and _strdup\(\)](#) — [Make a copy of a string in the heap](#)
- [strcmp\(\) and _strcmp\(\)](#) — [Compare strings, ignoring case](#)
- [strlwr\(\) and _strlwr\(\)](#) — [Make string all lowercase](#)
- [strnicmp\(\) and _strnicmp\(\)](#) — [Compare strings, ignoring case](#)
- [strnset\(\) and _strnset\(\)](#) — [Fill part or all of a string with any character](#)
- [strrev\(\) and _strrev\(\)](#) — [Reverse the order of characters in a string](#)
- [strset\(\) and _strset\(\)](#) — [Fill a string with any character](#)
- [strupr\(\) and _strupr\(\)](#) — [Make string all uppercase](#)
- [swab\(\) and _swab\(\)](#) — [Swap bytes in a word](#)
- [tell\(\) and _tell\(\)](#) — [Report the current position in a file](#)
- [tzset\(\) and _tzset\(\)](#) — [Set local time-zone values](#)
- [ultoa\(\) and _ultoa\(\)](#) — [Convert an unsigned long to an ASCII string](#)
- [umask\(\) and _umask\(\)](#) — [Set global file permission mask](#)
- [ungetch\(\) and _ungetch\(\)](#) — [Push a character back into the input buffer](#)
- [unlink\(\) and _unlink\(\)](#) — [Delete a file](#)
- [utime\(\) and _utime\(\)](#) — [Update a file's date/time stamp](#)
- [utoa\(\) and _utoa\(\)](#) — [Convert an unsigned int to an ASCII string](#)
- [vdmemcpy\(\)](#) — [Copy to a volatile destination \(ARC only\)](#)
- [vfscanf\(\) and _vfscanf\(\)](#) — [Read values from a file using var-arg macros](#)
- [vscanf\(\) and _vscanf\(\)](#) — [Read values from `stdin` using var-arg macros](#)
- [vsdmemcpy\(\)](#) — [Copy from a volatile source to a volatile destination \(ARC only\)](#)
- [vsMemcpy\(\)](#) — [Copy from a volatile source \(ARC only\)](#)

- [vsscanf\(\) and _vsscanf\(\)](#) — [Read values from a string using var-arg macros](#)
- [write\(\) and _write\(\)](#) — [Write data to a file, unbuffered](#)
- [y0\(\), y1\(\), yn\(\) and _y0\(\), _y1\(\), _yn\(\)](#) — [Bessel function of the second kind](#)

Appendix D — Reentrancy Status

In This Appendix

- [Reentrant Functions](#)
- [Not-Reentrant Functions](#)

NOTE For discussion of reentrancy issues, and definitions of the terms *Reentrant* and *Not-Reentrant*, see [Reentrancy](#) on page 25.

Reentrant Functions

- [abs\(\) — Calculate the absolute value of an integer](#)
- [abs\(\) — Calculate the absolute value of any arithmetic type](#)
- [acos\(\) — Arc cosine](#)
- [acosf\(\) — Arc cosine using single-precision floating point](#)
- [acosh\(\) and _acosh\(\), acosh\(\) and _acosh\(\) — Hyperbolic arc cosine](#)
- [alloca\(\) and _alloca\(\), alloca\(\) and _alloca\(\) — Allocate memory on the stack](#)
- [asctime\(\) and _asctime\(\) — Convert a struct tm to printable form](#)
- [asin\(\) — Arc sine](#)
- [asinf\(\) — Arc sine using single-precision floating point](#)
- [asinh\(\) and _asinh\(\), asinh\(\) and _asinh\(\) — Hyperbolic arc sine](#)
- [assert\(\) — Abort if an assertion is false](#)
- [atan\(\) — Arc tangent](#)
- [atan2\(\) — Arc tangent of the angle defined by a point](#)
- [atan2f\(\) — Arc tangent of the angle defined by a point, using single-precision floating point](#)
- [atanf\(\) — Arc tangent using single-precision floating point](#)
- [atanh\(\) and _atanh\(\), atanh\(\) and _atanh\(\) — Hyperbolic arc tangent](#)
- [atof\(\) — Convert a prefix of a string to floating point](#)
- [atoi\(\) — Convert a prefix of a string to an int](#)
- [atol\(\) — Convert a prefix of a string to a long int](#)
- [atold\(\) — Convert a prefix of a string to a long double](#)
- [bprintf\(\) — Print to a string of given size](#)
- [bsearch\(\) — Perform a binary search](#)
- [cabs\(\) and _cabs\(\), cabs\(\) and _cabs\(\) — Compute complex absolute value](#)
- [calloc\(\) — Dynamically allocate zero-initialized storage for objects](#)
- [ceil\(\) — Calculate smallest integer greater than or equal to a value](#)
- [ceilf\(\) — Calculate smallest integer greater than or equal to a value using single-precision floating point](#)
- [cos\(\) — Cosine](#)
- [cosf\(\) — Cosine using single-precision floating point](#)
- [cosh\(\) — Hyperbolic cosine](#)
- [coshf\(\) — Hyperbolic cosine using single-precision floating point](#)
- [crotl\(\) and _crotr\(\), crotl\(\) and _crotr\(\) — Rotate a char left or right](#)

- [dieetomsbin\(\) and _dieetomsbin\(\)](#), [dieetomsbin\(\) and _dieetomsbin\(\)](#) — [Convert IEEE double to Microsoft double](#)
- [difftime\(\)](#) — [Calculate difference between two times](#)
- [_disable\(\)](#) — [Disable interrupts](#)
- [div\(\)](#) — [Perform integer division with remainder](#)
- [div0\(\)](#) — [Replaceable handler for integer division by zero \(ARC only\)](#)
- [dmsbintoieee\(\)](#) and [_dmsbintoieee\(\)](#), [dmsbintoieee\(\)](#) and [_dmsbintoieee\(\)](#) — [Convert Microsoft double to IEEE double](#)
- [_enable\(\)](#) — [Enable interrupts](#)
- [exit\(\)](#) — [Terminate a program normally](#)
- [exp\(\)](#) — [Calculate exponential \(e to the power x\)](#)
- [expf\(\)](#) — [Calculate exponential \(e to the power x\) using single-precision floating point](#)
- [fabs\(\)](#) — [Calculate absolute value of a floating-point number using double precision](#)
- [fabsf\(\)](#) — [Calculate absolute value of a floating-point number using single precision](#)
- [feof\(\)](#) — [Test for end-of-file](#)
- [ferror\(\)](#) — [Test for a read or write error on a file](#)
- [fieetomsbin\(\)](#) and [_fieetomsbin\(\)](#), [fieetomsbin\(\)](#) and [_fieetomsbin\(\)](#) — [Convert IEEE float to Microsoft format](#)
- [fileno\(\)](#) and [_fileno\(\)](#), [fileno\(\)](#) and [_fileno\(\)](#) — [Get file handle currently associated with a stream](#)
- [_findclose\(\)](#) and [_findfirst\(\)](#) and [_findnext\(\)](#) — [Find a file whose name matches a specification](#)
- [floor\(\)](#) — [Calculate largest integer not greater than a value](#)
- [floorf\(\)](#) — [Calculate largest integer not greater than a value using single-precision floating point](#)
- [fmod\(\)](#) — [Return floating-point remainder](#)
- [fmodf\(\)](#) — [Return floating-point remainder using single precision](#)
- [fmsbintoieee\(\)](#) and [_fmsbintoieee\(\)](#), [fmsbintoieee\(\)](#) and [_fmsbintoieee\(\)](#) — [Convert Microsoft float to IEEE float](#)
- [frexp\(\)](#) — [Break a double into fraction and exponent](#)
- [frexpf\(\)](#) — [Break a float into fraction and exponent](#)
- [getpid\(\)](#) and [_getpid\(\)](#) — [Get unique ID of the calling process](#)
- [getpid\(\)](#) and [_getpid\(\)](#) — [Get unique ID of the calling process](#)
- [gmtime\(\)](#) and [_gmtime\(\)](#), [gmtime\(\)](#) and [_gmtime\(\)](#) — [Convert a time_t value to a struct tm, adjusted to UTC](#)
- [hypot\(\)](#) and [_hypot\(\)](#), [hypot\(\)](#) and [_hypot\(\)](#) — [Hypotenuse of a triangle](#)
- [_inline\(\)](#) — [Place bytes of code in line](#)

- [invalidate_dcache\(\)](#) — [Invalidate data cache \(ARC only\)](#)
- [invalidate_icache\(\)](#) — [Invalidate instruction cache \(ARC only\)](#)
- [isalnum\(\)](#) — [Test for alphanumeric character](#)
- [isalpha\(\)](#) — [Test for alphabetic character](#)
- [isascii\(\) and _isascii\(\)](#), [isascii\(\) and _isascii\(\)](#) — [Test for ASCII character](#)
- [isatty\(\) and _isatty\(\)](#), [isatty\(\) and _isatty\(\)](#) — [Identify character devices](#)
- [isctrl\(\)](#) — [Test for control character](#)
- [isdigit\(\)](#) — [Test for numeric character](#)
- [isgraph\(\)](#) — [Test for visible character](#)
- [islower\(\)](#) — [Test for lowercase alphabetic character](#)
- [isodigit\(\)](#) — [Test for octal numeric character](#)
- [isprint\(\)](#) — [Test for printable character](#)
- [ispunct\(\)](#) — [Test for punctuation character](#)
- [isspace\(\)](#) — [Test for whitespace character](#)
- [isupper\(\)](#) — [Test for uppercase alphabetic character](#)
- [isxdigit\(\)](#) — [Test for hexadecimal numeric character](#)
- [itoa\(\) and _itoa\(\)](#), [itoa\(\) and _itoa\(\)](#) — [Convert an int value to an ASCII string](#)
- [j0\(\), j1\(\), jn\(\) and _j0\(\), _j1\(\), _jn\(\)](#), [j0\(\), j1\(\), jn\(\) and _j0\(\), _j1\(\), _jn\(\)](#) — [Bessel function of the first kind](#)
- [labs\(\)](#) — [Calculate the absolute value of a long int](#)
- [ldexp\(\)](#) — [Multiply by a power of 2](#)
- [ldexpf\(\)](#) — [Multiply by a power of 2 using single-precision floating point](#)
- [ldiv\(\)](#) — [Perform long int division with remainder](#)
- [lfind\(\) and _lfind\(\)](#), [lfind\(\) and _lfind\(\)](#) — [Search a linear list for a matching value, without modifying the list](#)
- [log\(\)](#) — [Calculate natural logarithm \(base e\)](#)
- [log10\(\)](#) — [Calculate common logarithm \(base 10\)](#)
- [log10f\(\)](#) — [Calculate common logarithm \(base 10\) using single-precision floating point](#)
- [log2\(\)](#) and [log2\(\)](#), [log2\(\) and _log2\(\)](#) — [Calculate logarithm \(base 2\)](#)
- [logf\(\)](#) — [Calculate natural logarithm \(base e\) using single-precision floating point](#)
- [longjmp\(\)](#) — [Execute a non-local jump](#)
- [lrotl\(\) and _lrotl\(\)](#), [lrotl\(\) and _lrotl\(\)](#) — [Rotate a long int left or right](#)
- [lsearch\(\) and _lsearch\(\)](#), [lsearch\(\) and _lsearch\(\)](#) — [Search a linear list for a matching value, modifying if necessary](#)
- [lseek\(\) and _lseek\(\)](#) — [Change the position in a file](#)

- [`ltoa\(\)` and `_ltoa\(\)`, `ltoa\(\)` and `_ltoa\(\)`](#) — [Convert a long int value to an ASCII string](#)
- [`makepath\(\)`](#) — [Create a path string from components](#)
- [`max\(\)`](#) — [Determine maximum of any arithmetic type](#)
- [`mblen\(\)`](#) — [Determine length of a multibyte character](#)
- [`mbstowcs\(\)`](#) — [Convert a multibyte string to wchar_t array](#)
- [`mbtowc\(\)`](#) — [Convert a multibyte character to wchar_t code](#)
- [`memccpy\(\)` and `_memccpy\(\)`, `memccpy\(\)` and `_memccpy\(\)`](#) — [Copy memory until count or character](#)
- [`memchr\(\)`](#) — [Find a character in an area of memory](#)
- [`memcmp\(\)`](#) — [Compare two areas of memory](#)
- [`memcpy\(\)`](#) — [Copy from one place in memory to another](#)
- [`memicmp\(\)` and `_memicmp\(\)`, `memicmp\(\)` and `_memicmp\(\)`](#) — [Compare bytes ignoring case](#)
- [`memmove\(\)`](#) — [Copy memory nondestructively](#)
- [`memset\(\)`](#) — [Duplicate a character across an area of memory](#)
- [`min\(\)`](#) — [Determine minimum of any arithmetic type](#)
- [`mktimed\(\)`](#) — [Convert a struct tm to a time_t value](#)
- [`modf\(\)`](#) — [Break a double into integer and fractional parts](#)
- [`modff\(\)`](#) — [Break a float into integer and fractional parts](#)
- [`pow\(\)`](#) — [Raise a double to a power](#)
- [`powf\(\)`](#) — [Raise a float to a power](#)
- [`qsort\(\)`](#) — [Perform a quicksort](#)
- [`rmemcpy\(\)`](#) — [Copy from one place in memory to another](#)
- [`rotl\(\)` and `_rotr\(\)`, `rotl\(\)` and `_rotr\(\)`](#) — [Rotate an int left or right](#)
- [`rstrcpy\(\)`](#) — [Copy one string onto another](#)
- [`rstrncpy\(\)`](#) — [Copy a number of characters from one string into another](#)
- [`searchenv\(\)`](#) — [Search for a file using an environment variable](#)
- [`searchstr\(\)`](#) — [Search for a file in a list of directories](#)
- [`setjmp\(\)`](#) — [Save a reference to the current calling environment for a subsequent non-local jump](#)
- [`set_matherr\(\)`](#) — [User defined math error function](#)
- [`setmode\(\)`](#) — [Set stream mode \(`fmode`\) to text or binary](#)
- [`setvect1\(\)`](#) — [Set low-priority interrupt to jump to target \(ARC only\)](#)
- [`setvect2\(\)`](#) — [Set mid- or high-priority interrupt to jump to target \(ARC only\)](#)
- [`sin\(\)`](#) — [Sine](#)
- [`sinf\(\)`](#) — [Sine using single-precision floating point](#)

- [`sinh\(\)`](#) — [Hyperbolic sine](#)
- [`sinhf\(\)`](#) — [Hyperbolic sine using single-precision floating point](#)
- [`splitpath\(\)`](#) — [Split a pathname into its components](#)
- [`sprintf\(\)`](#) — [Print to a string](#)
- [`sqrt\(\)`](#) — [Square root](#)
- [`sqrtf\(\)`](#) — [Square root using single-precision floating point](#)
- [`_srotl\(\)` and `_srotr\(\)`, `_srotl\(\)` and `_srotr\(\)`](#) — [Rotate a short int left or right](#)
- [`sscanf\(\)`](#) — [Read values from a string](#)
- [`streat\(\)`](#) — [Concatenate two strings](#)
- [`strcats\(\)`](#) — [Concatenate multiple strings up to a limited number of characters](#)
- [`strchr\(\)`](#) — [Find the first occurrence of a character in a string](#)
- [`strcmp\(\)`](#) — [Compare one string to another](#)
- [`strcmpi\(\)` and `_strcmpi\(\)`, `stremp\(\)` and `_stremp\(\)`](#) — [Compare strings, ignoring case](#)
- [`strcoll\(\)`](#) — [Compare strings based on current locale](#)
- [`strcpy\(\)`](#) — [Copy the characters of one string into another](#)
- [`strcspn\(\)`](#) — [Determine the length of the prefix of a string not containing any characters from another string](#)
- [`strerror\(\)` and `_strerror\(\)`, `strerror\(\)` and `_strerror\(\)`](#) — [Map error code; append error-message string to user message](#)
- [`strftime\(\)`](#) — [Format time and date string](#)
- [`strcmp\(\)` and `_strcmp\(\)`, `stricmp\(\)` and `_stricmp\(\)`](#) — [Compare strings, ignoring case](#)
- [`strlen\(\)`](#) — [Determine the length of a string](#)
- [`strlwr\(\)` and `_strlwr\(\)`, `strlwr\(\)` and `_strlwr\(\)`](#) — [Make string all lowercase](#)
- [`strncat\(\)` and `_strncat\(\)`](#) — [Append characters of one string to another, up to some limit n](#)
- [`strncmp\(\)`](#) — [Compare characters of one string to characters of another](#)
- [`strncpy\(\)`](#) — [Copy a number of characters from one string into another](#)
- [`strnicmp\(\)` and `_strnicmp\(\)`, `strnicmp\(\)` and `_strnicmp\(\)`](#) — [Compare strings, ignoring case](#)
- [`strnset\(\)` and `_strnset\(\)`, `strnset\(\)` and `_strnset\(\)`](#) — [Fill part or all of a string with any character](#)
- [`strupr\(\)`](#) — [Find the first occurrence of any character of one string in another](#)
- [`strrchr\(\)`](#) — [Find the last occurrence of a character in a string](#)
- [`strrev\(\)` and `_strrev\(\)`, `strrev\(\)` and `_strrev\(\)`](#) — [Reverse the order of characters in a string](#)
- [`strset\(\)` and `_strset\(\)`, `strset\(\)` and `_strset\(\)`](#) — [Fill a string with any character](#)
- [`strspn\(\)`](#) — [Determine the length of the prefix of a string composed entirely of characters from another string](#)

- [strstr\(\) — Find the first occurrence of one string within another](#)
- [strtod\(\) — Convert a string to a double](#)
- [strtol\(\) — Convert a string to a long](#)
- [strtoul\(\) — Convert a string to an unsigned long](#)
- [strupr\(\) and _strupr\(\), strupr\(\) and _strupr\(\) — Make string all uppercase](#)
- [strxfrm\(\) — Transform string to locale-independent form](#)
- [swab\(\) and _swab\(\), swab\(\) and _swab\(\) — Swap bytes in a word](#)
- [tan\(\) — Tangent](#)
- [tanf\(\) — Tangent using single-precision floating point](#)
- [tanh\(\) — Hyperbolic tangent](#)
- [tanhf\(\) — Hyperbolic tangent using single-precision floating point](#)
- [time\(\) — Get the current time and date](#)
- [tolower\(\) — Convert to lowercase](#)
- [tolower\(\) — Convert to lowercase without testing for uppercase](#)
- [toupper\(\) — Convert to uppercase](#)
- [toupper\(\) — Convert to uppercase without testing for lowercase](#)
- [udiv0\(\) — Replaceable handler for unsigned integer division by zero \(ARC only\)](#)
- [ultoa\(\) and _ultoa\(\), ultoa\(\) and _ultoa\(\) — Convert an unsigned long to an ASCII string](#)
- [utoa\(\) and _utoa\(\), utoa\(\) and _utoa\(\) — Convert an unsigned int to an ASCII string](#)
- [va_arg\(\) — Get the next argument in a variable series of arguments](#)
- [va_end\(\) — Terminate va_list\(\) processing](#)
- [vbprintf\(\) — Print to a string of given size using var-arg macros](#)
- [vdmemcpy\(\) — Copy to a volatile destination \(ARC only\)](#)
- [vsdmemcpy\(\) — Copy from a volatile source to a volatile destination \(ARC only\)](#)
- [vmemset\(\) — Duplicate a character to a volatile destination \(ARC only\)](#)
- [vsmemcpy\(\) — Copy from a volatile source \(ARC only\)](#)
- [vsprintf\(\) — Print to a string using var-arg macros](#)
- [vsscanf\(\) and _vsscanf\(\), vsscanf\(\) and _vsscanf\(\) — Read values from a string using var-arg macros](#)
- [wctomb\(\) — Convert a wchar_t array to a multibyte string](#)
- [wctomb\(\) — Convert a wchar_t code to a multibyte character](#)
- [y0\(\), y1\(\), yn\(\) and _y0\(\), _y1\(\), _yn\(\), y0\(\), y1\(\), yn\(\) and _y0\(\), _y1\(\), _yn\(\) — Bessel function of the second kind](#)

Not-Reentrant Functions

- [abort\(\)](#) — [Terminate a program abnormally](#)
- [access\(\) and _access\(\)](#), [access\(\) and _access\(\)](#) — [Test access rights of a file](#)
- [asctime\(\) and _asctime\(\)](#) — [Convert a struct tm to printable form](#)
- [atexit\(\)](#) — [Register a function for execution at program termination](#)
- [cgets\(\) and _cgets\(\)](#), [cgets\(\) and _cgets\(\)](#) — [Read a line of text \(string\) from the keyboard](#)
- [chdir\(\) and _chdir\(\)](#), [chdir\(\) and _chdir\(\)](#) — [Change the current working directory](#)
- [chmod\(\) and _chmod\(\)](#), [chmod\(\) and _chmod\(\)](#) — [Alter the permission setting for a file](#)
- [chsize\(\) and _chsize\(\)](#), [chsize\(\) and _chsize\(\)](#) — [Extend or truncate the length of a file](#)
- [clearerr\(\)](#) — [Clear end-of-file and error flags](#)
- [clock\(\)](#) — [Report elapsed processor time](#)
- [close\(\) and _close\(\)](#), [close\(\) and _close\(\)](#) — [Close a file handle](#)
- [closedir\(\)](#) — [Close a POSIX directory stream](#)
- [cprintf\(\) and _cprintf\(\)](#), [cprintf\(\) and _cprintf\(\)](#) — [Print to screen](#)
- [cputs\(\) and _cputs\(\)](#), [cputs\(\) and _cputs\(\)](#) — [Write a string to the screen](#)
- [cscanf\(\) and _cscanf\(\)](#), [cscanf\(\) and _cscanf\(\)](#) — [Read from standard input](#)
- [ctime\(\) and _ctime\(\)](#), [ctime\(\) and _ctime\(\)](#) — [Convert a time_t value to printable form](#)
- [dup\(\) and _dup\(\)](#), [dup\(\) and _dup\(\)](#) — [Duplicate a file handle](#)
- [dup2\(\) and _dup2\(\)](#), [dup2\(\) and _dup2\(\)](#) — [Duplicate one file handle into another](#)
- [ecvt\(\) and _ecvt\(\)](#), [ecvt\(\) and _ecvt\(\)](#) — [Convert a floating-point number to a character string](#)
- [eof\(\) and _eof\(\)](#), [eof\(\) and _eof\(\)](#) — [Test a file handle for end-of-file status](#)
- [exit\(\)](#) — [Terminate a program immediately](#)
- [fclose\(\)](#) — [Close a file](#)
- [fcloseall\(\) and _fcloseall\(\)](#), [fcloseall\(\) and _fcloseall\(\)](#) — [Close all files](#)
- [fcvt\(\) and _fcvt\(\)](#), [fcvt\(\) and _fcvt\(\)](#) — [Convert a floating-point number to a character string](#)
- [fdopen\(\) and _fdopen\(\)](#), [fdopen\(\) and _fdopen\(\)](#) — [Associate a stream with an open handle](#)
- [fflush\(\)](#) — [Flush a file's buffer](#)
- [fgetc\(\)](#) — [Read \(get\) a character from a file](#)
- [fgetchar\(\) and _fgetchar\(\)](#), [fgetchar\(\) and _fgetchar\(\)](#) — [Get a character from standard input](#)
- [fgetpos\(\)](#) — [Determine file position](#)
- [fgets\(\)](#) — [Read a line of text \(string\) from a file](#)
- [filelength\(\) and _filelength\(\)](#), [filelength\(\) and _filelength\(\)](#) — [Determine length of an open file](#)

- [findclose\(\)](#) and [findfirst\(\)](#) and [findnext\(\)](#), [findclose\(\)](#) and [findfirst\(\)](#) and [findnext\(\)](#), [findclose\(\)](#) and [findfirst\(\)](#) and [findnext\(\)](#) — [Find a file whose name matches a specification](#)
- [flushall\(\)](#) and [flushall\(\)](#), [flushall\(\)](#) and [flushall\(\)](#) — [Clear input buffers and flush output buffers](#)
- [fopen\(\)](#) — [Open a file](#)
- [FP_OFFSET\(\)](#), [FP_SEG\(\)](#) and [FP_OFFSET\(\)](#), [FP_SEG\(\)](#), [FP_OFFSET\(\)](#), [FP_SEG\(\)](#) and [FP_OFFSET\(\)](#), [FP_SEG\(\)](#) — [Get offset and segment of a pointer](#)
- [fprintf\(\)](#) — [Print to a file](#)
- [fputc\(\)](#) — [Write a character to a file](#)
- [fputchar\(\)](#) and [_fputchar\(\)](#), [fputchar\(\)](#) and [_fputchar\(\)](#) — [Write a character to standard output](#)
- [fputs\(\)](#) — [Write a string to a file](#)
- [fread\(\)](#) — [Read from a file](#)
- [free\(\)](#) — [Release storage allocated by calloc\(\), malloc\(\), or realloc\(\)](#)
- [freopen\(\)](#) — [Open a file using an existing FILE variable](#)
- [fscanf\(\)](#) — [Read values from a file](#)
- [fseek\(\)](#) — [Move file pointer to new location in file](#)
- [fsetpos\(\)](#) — [Set file position](#)
- [fseek\(\)](#) — [Move file pointer to new location in file](#)
- [fsetpos\(\)](#) — [Set file position](#)
- [ftell\(\)](#) — [Determine current value of a file pointer](#)
- [ftime\(\)](#) and [_ftime\(\)](#), [ftime\(\)](#) and [_ftime\(\)](#) — [Get current time values](#)
- [fullpath\(\)](#) — [Convert relative pathname to absolute pathname](#)
- [fwrite\(\)](#) — [Write to a file](#)
- [gcvt\(\)](#) and [_gcvt\(\)](#) — [Convert a floating-point number to a character string](#)
- [getc\(\)](#) — [Get a character from a file](#)
- [getchar\(\)](#) — [Get a character from standard input](#)
- [getcwd\(\)](#) and [_getcwd\(\)](#), [getcwd\(\)](#) and [_getcwd\(\)](#) — [Get full pathname of the current working directory](#)
- [getenv\(\)](#) — [Get the value of an environment variable](#)
- [gets\(\)](#) — [Read a line of text \(string\) from standard input](#)
- [getw\(\)](#) and [_getw\(\)](#), [getw\(\)](#) and [_getw\(\)](#) — [Get an integer from a file](#)
- [heapchk\(\)](#) — [Check heap for consistency](#)
- [heapset\(\)](#) — [Fill free memory locations in the heap](#)
- [heapwalk\(\)](#) — [Walk through the heap](#)
- [isatty\(\)](#) and [_isatty\(\)](#), [isatty\(\)](#) and [_isatty\(\)](#) — [Identify character devices](#)

- [ldecvt\(\) — Convert a long double to a character string](#)
- [ldfcvt\(\) — Convert a long double to a character string](#)
- [localeconv\(\) — Query current locale for numeric format](#)
- [localtime\(\) and localtime\(\), localtime\(\) and localtime\(\) — Convert a time_t value to a struct tm](#)
- [lseek\(\) and lseek\(\) — Change the position in a file](#)
- [malloc\(\) — Dynamically allocate uninitialized storage](#)
- [matherr\(\), matherrx\(\) and matherr\(\), matherrx\(\), matherr\(\), matherrx\(\) and matherr\(\), matherrx\(\) — Provide standard error reporting for math functions](#)
- [mkdir\(\) and mkdir\(\), mkdir\(\) and mkdir\(\) — Create a new directory](#)
- [mktemp\(\) and mktemp\(\), mktemp\(\) and mktemp\(\) — Create a unique file name \(does not create or open files\)](#)
- [msize\(\) — Get the size of a memory block in the heap](#)
- [open\(\) and open\(\), open\(\) and open\(\) — Open a file handle with a pathname](#)
- [opendir\(\) — Open a POSIX directory stream](#)
- [perror\(\) — Print an error message](#)
- [printf\(\) — Print to stdout](#)
- [putc\(\) — Write a character to a file](#)
- [putch\(\) and putch\(\), putch\(\) and putch\(\) — Write a character to the screen](#)
- [putchar\(\) — Write a character to standard output](#)
- [putenv\(\) and putenv\(\) — Add or modify an environment variable](#)
- [puts\(\) — Write a string to standard output](#)
- [putw\(\) and putw\(\), putw\(\) and putw\(\) — Write an integer to a file](#)
- [raise\(\) — Raise a signal](#)
- [rand\(\) — Generate a pseudo-random number](#)
- [read\(\) and read\(\), read\(\) and read\(\) — Read data from a file using a handle](#)
- [readdir\(\) — Read a POSIX directory stream](#)
- [realloc\(\) — Reallocate storage allocated by calloc\(\) or malloc\(\)](#)
- [remove\(\) — Delete a file](#)
- [rename\(\) — Change the name of a file](#)
- [rewind\(\) — Seek to the beginning of a file](#)
- [rewinddir\(\) — Reset a POSIX directory stream](#)
- [rmdir\(\) and rmdir\(\), rmdir\(\) and rmdir\(\) — Remove a directory](#)
- [scanf\(\) — Read values from stdin](#)
- [setbuf\(\) — Specify a buffer for a stream](#)

- [setlocale\(\) — Set the current locale](#)
- [setvbuf\(\) — Control file buffering](#)
- [signal\(\) — Set up a signal handler](#)
- [sleep\(\) — Temporarily suspend program execution](#)
- [spawn*\(\) and _spawn*\(\), spawn*\(\) and _spawn*\(\) — Execute a child process](#)
- [srand\(\) — Seed the pseudo-random number generator](#)
- [stat\(\) and _stat\(\), stat\(\) and _stat\(\) — Get information about a file or directory](#)
- [strdate\(\) — Convert the current date to a string](#)
- [strdup\(\) and _strdup\(\), strdup\(\) and _strdup\(\) — Make a copy of a string in the heap](#)
- [strftime\(\) — Convert current time of day to a string](#)
- [strtok\(\) — Divide a string into tokens](#)
- [system\(\) — Pass a command to the operating system](#)
- [tell\(\) and _tell\(\), tell\(\) and _tell\(\) — Report the current position in a file](#)
- [tmpfile\(\) and _tmpfile\(\), tmpfile\(\) and _tmpfile\(\) — Create a temporary file](#)
- [tempnam\(\) — Generate a temporary file name in a directory](#)
- [tmpnam\(\) — Generate a string to be used as a temporary file name](#)
- [tzset\(\) and _tzset\(\), tzset\(\) and _tzset\(\) — Set local time-zone values](#)
- [umask\(\) and _umask\(\), umask\(\) and _umask\(\) — Set global file permission mask](#)
- [ungetc\(\) — Push a character back into an input stream](#)
- [ungetch\(\) and _ungetch\(\), ungetch\(\) and _ungetch\(\) — Push a character back into the input buffer](#)
- [unlink\(\) and _unlink\(\), unlink\(\) and _unlink\(\) — Delete a file](#)
- [utime\(\) and _utime\(\), utime\(\) and _utime\(\) — Update a file's date/time stamp](#)
- [vfprintf\(\) — Print to a file using var-arg macros](#)
- [vfscanf\(\) and _vfscanf\(\), vfscanf\(\) and _vfscanf\(\) — Read values from a file using var-arg macros](#)
- [vprintf\(\) — Print to **stdout** using var-arg macros](#)
- [vscanf\(\) and _vscanf\(\), vscanf\(\) and _vscanf\(\) — Read values from **stdin** using var-arg macros](#)
- [write\(\) and _write\(\), write\(\) and _write\(\) — Write data to a file, unbuffered](#)

Appendix E — Function Families

In This Appendix

- [ARC-Only Functions](#)
- [The *printf Family of Functions](#)
- [The *scanf Family of Functions](#)
- [Transcendental Math Functions](#)
- [Conversion Functions](#)
- [I/O Functions](#)
- [File-Related Functions](#)
- [String-Handling Functions](#)
- [Memory Allocation and Deallocation Functions](#)
- [Time-Related Functions](#)

ARC-Only Functions

<u>div0()</u>	Replaceable handler for integer division by zero (ARC only)
<u>invalidate_dcache()</u>	Invalidate data cache (ARC only)
<u>invalidate_icache()</u>	Invalidate instruction cache (ARC only)
<u>setvect1()</u>	Set low-priority interrupt to jump to target (ARC only)
<u>setvect2()</u>	Set mid- or high-priority interrupt to jump to target (ARC only)
<u>udiv0()</u>	Replaceable handler for unsigned integer division by zero (ARC only)
<u>vdmemcpy()</u>	Copy to a volatile destination (ARC only)
<u>vmemset()</u>	Duplicate a character to a volatile destination (ARC only)
<u>vsdmemcpy()</u>	Copy from a volatile source to a volatile destination (ARC only)
<u>vsmemcpy()</u>	Copy from a volatile source (ARC only)

The *printf Family of Functions

<u>bprintf()</u>	Print to a string of given size
<u>cprintf() and _cprintf()</u>	Print to screen
<u>fprintf()</u>	Print to a file
<u>printf()</u>	Print to <code>stdout</code>
<u>sprintf()</u>	Print to a string
<u>vbprintf()</u>	Print to a string of given size using var-arg macros
<u>vfprintf()</u>	Print to a file using var-arg macros
<u>vprintf()</u>	Print to <code>stdout</code> using var-arg macros
<u>vsprintf()</u>	Print to a string using var-arg macros

The *scanf Family of Functions

<u>cscanf() and _cscanf()</u>	Read from standard input
<u>fscanf()</u>	Read values from a file
<u>scanf()</u>	Read values from <code>stdin</code>
<u>sscanf()</u>	Read values from a string
<u>vfscanf() and _vfscanf()</u>	Read values from a file using var-arg macros
<u>vscanf() and _vscanf()</u>	Read values from <code>stdin</code> using var-arg macros
<u>vsscanf() and _vsscanf()</u>	Read values from a string using var-arg macros

Transcendental Math Functions

<u>cos()</u>	Cosine
<u>cosf()</u>	Cosine using single-precision floating point
<u>sin()</u>	Sine
<u>sinf()</u>	Sine using single-precision floating point
<u>tan()</u>	Tangent

<u>tanf()</u>	<u>Tangent using single-precision floating point</u>
<u>acos()</u>	<u>Arc cosine</u>
<u>acosf()</u>	<u>Arc cosine using single-precision floating point</u>
<u>asin()</u>	<u>Arc sine</u>
<u>asinf()</u>	<u>Arc sine using single-precision floating point</u>
<u>atan()</u>	<u>Arc tangent</u>
<u>atanf()</u>	<u>Arc tangent using single-precision floating point</u>
<u>atan2()</u>	<u>Arc tangent of the angle defined by a point</u>
<u>atan2f()</u>	<u>Arc tangent of the angle defined by a point, using single-precision floating point</u>
<u>cosh()</u>	<u>Hyperbolic cosine</u>
<u>coshf()</u>	<u>Hyperbolic cosine using single-precision floating point</u>
<u>sinh()</u>	<u>Hyperbolic sine</u>
<u>sinhf()</u>	<u>Hyperbolic sine using single-precision floating point</u>
<u>tanh()</u>	<u>Hyperbolic tangent</u>
<u>tanhf()</u>	<u>Hyperbolic tangent using single-precision floating point</u>
<u>acosh() and _acosh()</u>	<u>Hyperbolic arc cosine</u>
<u>asinh() and _asinh()</u>	<u>Hyperbolic arc sine</u>
<u>atanh() and _atanh()</u>	<u>Hyperbolic arc tangent</u>
<u>hypot() and _hypot()</u>	<u>Hypotenuse of a triangle</u>

Conversion Functions

The functions listed in this section are grouped into the following categories:

- [Case Conversions](#)
- [String-to-Numeric Conversions](#)
- [Numeric-to-String Conversions](#)
- [IEEE/Microsoft Binary Conversions](#)
- [Multibyte/Wide-Character Conversions](#)
- [int and long int Conversions](#)
- [unsigned int and unsigned long Conversions](#)
- [double and long double Conversions](#)
- [float Conversions](#)

Case Conversions

<u>strlwr() and _strlwr()</u>	<u>Make string all lowercase</u>
<u>strupr() and _strupr()</u>	<u>Make string all uppercase</u>

<u>tolower()</u>	Convert to lowercase
<u>tolower()</u>	Convert to lowercase without testing for uppercase
<u>toupper()</u>	Convert to uppercase
<u>toupper()</u>	Convert to uppercase without testing for lowercase

String-to-Numeric Conversions

<u>atof()</u>	Convert a prefix of a string to floating point
<u>atoi()</u>	Convert a prefix of a string to an int
<u>atol()</u>	Convert a prefix of a string to a long int
<u>atold()</u>	Convert a prefix of a string to a long double
<u>strtod()</u>	Convert a string to a double
<u>strtol()</u>	Convert a string to a long
<u>strtoul()</u>	Convert a string to an unsigned long

Numeric-to-String Conversions

<u>ecvt() and _ecvt()</u>	Convert a floating-point number to a character string
<u>fcvt() and _fcvt()</u>	Convert a floating-point number to a character string
<u>gcvt() and _gcvt()</u>	Convert a floating-point number to a character string
<u>itoa() and _itoa()</u>	Convert an int value to an ASCII string
<u>ltoa() and _ltoa()</u>	Convert a long int value to an ASCII string
<u>ultoa() and _ultoa()</u>	Convert an unsigned long to an ASCII string
<u>utoa() and _utoa()</u>	Convert an unsigned int to an ASCII string

IEEE/Microsoft Binary Conversions

<u>dieeeetomsb()</u> and <u>dieeetomsbin()</u>	Convert IEEE double to Microsoft double
<u>dmsbintoieee()</u> and <u>dmsbintoieeee()</u>	Convert Microsoft double to IEEE double
<u>fieetomsb()</u> and <u>fieetomsbin()</u>	Convert IEEE float to Microsoft format
<u>fmsbintoieee()</u> and <u>fmsbintoieeee()</u>	Convert Microsoft float to IEEE float

Multibyte/Wide-Character Conversions

<u>mblen()</u>	Determine length of a multibyte character
<u>mbstowcs()</u>	Convert a multibyte string to wchar_t array
<u>mbtowc()</u>	Convert a multibyte character to wchar_t code
<u>wctomb()</u>	Convert a wchar_t array to a multibyte string
<u>wctomb()</u>	Convert a wchar_t code to a multibyte character

int and long int Conversions

<u>atoi()</u>	Convert a prefix of a string to an int
<u>atol()</u>	Convert a prefix of a string to a long int
<u>itoa() and _itoa()</u>	Convert an int value to an ASCII string

<u>itoa() and _itoa()</u>	Convert a long int value to an ASCII string
<u>strtol()</u>	Convert a string to a long

unsigned int and unsigned long Conversions

<u>strtoul()</u>	Convert a string to an unsigned long
<u>ultoa() and _ultoa()</u>	Convert an unsigned long to an ASCII string
<u>utoa() and _utoa()</u>	Convert an unsigned int to an ASCII string

double and long double Conversions

<u>atold()</u>	Convert a prefix of a string to a long double
<u>ldecvt()</u>	Convert a long double to a character string
<u>ldfcvt()</u>	Convert a long double to a character string
<u>strtod()</u>	Convert a string to a double

float Conversions

<u>atof()</u>	Convert a prefix of a string to floating point
<u>ecvt() and _ecvt()</u>	Convert a floating-point number to a character string
<u>fcvt() and _fcvt()</u>	Convert a floating-point number to a character string
<u>gcvt() and _gcvt()</u>	Convert a floating-point number to a character string

I/O Functions

The functions listed in this section are grouped into the following categories:

- [Standard I/O](#)
- [Keyboard/console I/O](#)
- [Character I/O](#)
- [String I/O](#)
- [File I/O](#)

Standard I/O

<u>cscanf() and _cscanf()</u>	Read from standard input
<u>fgetchar() and _fgetchar()</u>	Get a character from standard input
<u>fputchar() and _fputchar()</u>	Write a character to standard output
<u>getchar()</u>	Get a character from standard input
<u>gets()</u>	Read a line of text (string) from standard input
<u>putchar()</u>	Write a character to standard output
<u>puts()</u>	Write a string to standard output
<u>vprintf()</u>	Print to <code>stdout</code> using var-arg macros
<u>vscanf() and _vscanf()</u>	Read values from <code>stdin</code> using var-arg macros

Keyboard/console I/O

<u>cgets() and _cgets()</u>	Read a line of text (string) from the keyboard
<u>cprintf() and _cprintf()</u>	Print to screen
<u>cputs() and _cputs()</u>	Write a string to the screen
<u>putch() and _putch()</u>	Write a character to the screen

Character I/O

<u>getc()</u>	Get a character from a file
<u>getchar()</u>	Get a character from standard input
<u>fgetc()</u>	Read (get) a character from a file
<u>fgetchar() and _fgetchar()</u>	Get a character from standard input
<u>fputc()</u>	Write a character to a file
<u>fputchar() and _fputchar()</u>	Write a character to standard output
<u>putc()</u>	Write a character to a file
<u>putch() and _putch()</u>	Write a character to the screen
<u>putchar()</u>	Write a character to standard output
<u>ungetc()</u>	Push a character back into an input stream

String I/O

<u>cgets() and _cgets()</u>	Read a line of text (string) from the keyboard
<u>fgets()</u>	Read a line of text (string) from a file
<u>gets()</u>	Read a line of text (string) from standard input
<u>sscanf()</u>	Read values from a string
<u>vscanf() and _vscanf()</u>	Read values from a string using var-arg macros
 <u>cputs() and _cputs()</u>	Write a string to the screen
<u>fputs()</u>	Write a string to a file
<u>puts()</u>	Write a string to standard output
 <u>bprintf()</u>	Print to a string of given size
<u>sprintf()</u>	Print to a string
<u>vbprintf()</u>	Print to a string of given size using var-arg macros
<u>vsprintf()</u>	Print to a string using var-arg macros

File I/O

<u>clearerr()</u>	Clear end-of-file and error flags
<u>eof() and _eof()</u>	Test a file handle for end-of-file status
<u>feof()</u>	Test for end-of-file
<u>ferror()</u>	Test for a read or write error on a file

<u>fgetc()</u>	<u>Read (get) a character from a file</u>
<u>fgets()</u>	<u>Read a line of text (string) from a file</u>
<u>fread()</u>	<u>Read from a file</u>
<u>fscanf()</u>	<u>Read values from a file</u>
<u>getc()</u>	<u>Get a character from a file</u>
<u>getw() and _getw()</u>	<u>Get an integer from a file</u>
<u>read() and _read()</u>	<u>Read data from a file using a handle</u>
<u>vfscanf() and _vfscanf()</u>	<u>Read values from a file using var-arg macros</u>
<u>fprintf()</u>	<u>Print to a file</u>
<u>printf()</u>	<u>Print to <code>stdout</code></u>
<u>vfprintf()</u>	<u>Print to a file using var-arg macros</u>
<u>fputc()</u>	<u>Write a character to a file</u>
<u>fputs()</u>	<u>Write a string to a file</u>
<u>fwrite()</u>	<u>Write to a file</u>
<u>putc()</u>	<u>Write a character to a file</u>
<u>putw() and _putw()</u>	<u>Write an integer to a file</u>
<u>write() and _write()</u>	<u>Write data to a file, unbuffered</u>

File-Related Functions

All the functions listed in this section are related to manipulating files. These functions are grouped into the following categories:

- [File-Access Functions](#)
- [Get Information about a File, Directory, Handle, or Stream](#)
- [Modify an Existing File or Directory](#)
- [Find an Existing File](#)
- [Pathname Functions](#)
- [Create a File, a File Name, or a Directory](#)
- [Duplicate a File Handle](#)
- [Open a File](#)
- [Read from a File](#)
- [Write or Print to a File](#)
- [File-Position Functions \(and Macros\)](#)
- [File-Buffering Functions](#)
- [Close a File or Directory Stream](#)
- [Delete a File or Directory](#)

File-Access Functions

<u>access() and _access()</u>	Test access rights of a file
<u>umask() and _umask()</u>	Set global file permission mask

Get Information about a File, Directory, Handle, or Stream

<u>access()and _access()</u>	Test access rights of a file
<u>eof() and _eof()</u>	Test a file handle for end-of-file status
<u>feof()</u>	Test for end-of-file
<u>ferror()</u>	Test for a read or write error on a file
<u>fileno()and _fileno()</u>	Get file handle currently associated with a stream
<u>filelength() and _filelength()</u>	Determine length of an open file
<u>tell()</u>	Determine current value of a file pointer
<u>stat() and _stat()</u>	Get information about a file or directory
<u>tell() and _tell()</u>	Report the current position in a file

Modify an Existing File or Directory

<u>chmod() and _chmod()</u>	Alter the permission setting for a file
<u>chsize()and _chsize()</u>	Extend or truncate the length of a file
<u>clearerr()</u>	Clear end-of-file and error flags
<u>rename()</u>	Change the name of a file
<u>setmode()</u>	Set stream mode (<u>fmode</u>) to text or binary

Find an Existing File

<u>findclose() and _findfirst() and _findnext()</u>, <u>findclose() and _findfirst() and _findnext()</u>, <u>findclose() and _findfirst() and _findnext()</u>	Find a file whose name matches a specification
<u>searchenv()</u>	Search for a file using an environment variable
<u>searchstr()</u>	Search for a file in a list of directories

Pathname Functions

<u>fullpath()</u>	Convert relative pathname to absolute pathname
<u>makepath()</u>	Create a path string from components
<u>splitpath()</u>	Split a pathname into its components

Create a File, a File Name, or a Directory

<u>creat() and _creat()</u>	Create an empty file
<u>mkdir()and _mkdir()</u>	Create a new directory
<u>mktemp() and _mktemp()</u>	Create a unique file name (does not create or open files)
<u>tempnam()</u>	Generate a temporary file name in a directory

<u>tmpfile() and _tmpfile()</u>	Create a temporary file
<u>tmpnam()</u>	Generate a string to be used as a temporary file name

Duplicate a File Handle

<u>dup() and _dup()</u>	Duplicate a file handle
<u>dup2() and _dup2()</u>	Duplicate one file handle into another

Open a File

<u>fdopen() and _fdopen()</u>	Associate a stream with an open handle
<u>fopen()</u>	Open a file
<u>freopen()</u>	Open a file using an existing FILE variable
<u>open() and _open()</u>	Open a file handle with a pathname

Read from a File

<u>fgetc()</u>	Read (get) a character from a file
<u>fgets()</u>	Read a line of text (string) from a file
<u>fread()</u>	Read from a file
<u>fscanf()</u>	Read values from a file
<u>getc()</u>	Get a character from a file
<u>getw() and _getw()</u>	Get an integer from a file
<u>read() and _read()</u>	Read data from a file using a handle
<u>vfscanf() and _vfscanf()</u>	Read values from a file using var-arg macros

Write or Print to a File

<u>fprintf()</u>	Print to a file
<u>fputc()</u>	Write a character to a file
<u>fputs()</u>	Write a string to a file
<u>fwrite()</u>	Write to a file
<u>putc()</u>	Write a character to a file
<u>putw() and _putw()</u>	Write an integer to a file
<u>vfprintf()</u>	Print to a file using var-arg macros
<u>write() and _write()</u>	Write data to a file, unbuffered

File-Position Functions (and Macros)

<u>fgetpos()</u>	Determine file position
<u>fseek()</u>	Move file pointer to new location in file
<u>fsetpos()</u>	Set file position
<u>lseek() and _lseek()</u>	Change the position in a file
<u>rewind()</u>	Seek to the beginning of a file
SEEK_CUR and SEEK_END and SEEK_SET, SEEK_CUR and SEEK_END and SEEK_SET, SEEK_CUR and SEEK_END and SEEK_SET	Macros used to specify position in file

File-Buffering Functions

<u>fflush()</u>	Flush a file's buffer
<u>flushall() and _flushall()</u>	Clear input buffers and flush output buffers
<u>setbuf()</u>	Specify a buffer for a stream
<u>setvbuf()</u>	Control file buffering

Close a File or Directory Stream

<u>close() and _close()</u>	Close a file handle
<u>closedir()</u>	Close a POSIX directory stream
<u>fclose()</u>	Close a file
<u>fcloseall() and _fcloseall()</u>	Close all files

Delete a File or Directory

<u>remove()</u>	Delete a file
<u>rmdir() and _rmdir()</u>	Remove a directory
<u>unlink() and _unlink()</u>	Delete a file

String-Handling Functions

All the functions listed in this section are related to using strings. These functions are grouped into the following categories:

- [Comparing Strings](#)
- [Concatenating or Appending Strings](#)
- [Converting, Transforming, or Formatting Strings](#)
- [Converting Other Data to Strings](#)
- [Copying Strings](#)
- [Determining String Length](#)
- [Filling Strings](#)
- [Finding Characters in Strings](#)

For more categories of string-related functions, see the following:

- [String-to-Numeric Conversions](#) (part of [Conversion Functions](#) on page 425)
- [Numeric-to-String Conversions](#) (part of [Conversion Functions](#) on page 425)
- [String I/O](#) (part of [I/O Functions](#) on page 427)

Comparing Strings

<u>strcmp()</u>	Compare one string to another
<u>strcmpi() and _strcmpi()</u>	Compare strings, ignoring case
<u>strcoll()</u>	Compare strings based on current locale
<u>strcmp() and _strcmp()</u>	Compare strings, ignoring case

<u>strncmp()</u>	<u>Compare characters of one string to characters of another</u>
<u>strnicmp() and _strnicmp()</u>	<u>Compare strings, ignoring case</u>

Concatenating or Appending Strings

<u>streat()</u>	<u>Concatenate two strings</u>
<u>streats()</u>	<u>Concatenate multiple strings up to a limited number of characters</u>
<u>strerror() and _strerror()</u>	<u>Map error code; append error-message string to user message</u>
<u>strncat() and _strncat()</u>	<u>Append characters of one string to another, up to some limit n</u>

Converting, Transforming, or Formatting Strings

<u>strftime()</u>	<u>Format time and date string</u>
<u>strlwr() and _strlwr()</u>	<u>Make string all lowercase</u>
<u>strrev() and _strrev()</u>	<u>Reverse the order of characters in a string</u>
<u>strtod()</u>	<u>Convert a string to a double</u>
<u>strtok()</u>	<u>Divide a string into tokens</u>
<u>strtol()</u>	<u>Convert a string to a long</u>
<u>strtoul()</u>	<u>Convert a string to an unsigned long</u>
<u>strupr() and _strupr()</u>	<u>Make string all uppercase</u>
<u>strxfrm()</u>	<u>Transform string to locale-independent form</u>

Converting Other Data to Strings

<u>strdate()</u>	<u>Convert the current date to a string</u>
<u>strtime()</u>	<u>Convert current time of day to a string</u>

Copying Strings

<u>rstrcpy()</u>	<u>Copy one string onto another</u>
<u>rstrncpy()</u>	<u>Copy a number of characters from one string into another</u>
<u>strcpy()</u>	<u>Copy the characters of one string into another</u>
<u>strdup() and _strdup()</u>	<u>Make a copy of a string in the heap</u>
<u>strncpy()</u>	<u>Copy a number of characters from one string into another</u>
<u>swab() and _swab()</u>	<u>Swap bytes in a word</u>

Determining String Length

<u>strcspn()</u>	<u>Determine the length of the prefix of a string not containing any characters from another string</u>
<u>strlen()</u>	<u>Determine the length of a string</u>
<u>strspn()</u>	<u>Determine the length of the prefix of a string composed entirely of characters from another string</u>

Filling Strings

<u>strnset() and _strnset()</u>	<u>Fill part or all of a string with any character</u>
<u>strset() and _strset()</u>	<u>Fill a string with any character</u>

Finding Characters in Strings

<u>strchr()</u>	Find the first occurrence of a character in a string
<u>strpbrk()</u>	Find the first occurrence of any character of one string in another
<u> strrchr()</u>	Find the last occurrence of a character in a string
<u>strstr()</u>	Find the first occurrence of one string within another

Memory Allocation and Deallocation Functions

<u>alloca() and _alloca()</u>	Allocate memory on the stack
<u>calloc()</u>	Dynamically allocate zero-initialized storage for objects
<u>free()</u>	Release storage allocated by calloc(), malloc(), or realloc()
<u>malloc()</u>	Dynamically allocate uninitialized storage
<u>realloc()</u>	Reallocate storage allocated by calloc() or malloc()

Time-Related Functions

<u>asctime() and _asctime()</u>	Convert a struct tm to printable form
<u>clock()</u>	Report elapsed processor time
<u>ctime() and _ctime()</u>	Convert a time_t value to printable form
<u>difftime()</u>	Calculate difference between two times
<u>ftime() and _ftime()</u>	Get current time values
<u>gmtime() and _gmtime()</u>	Convert a time_t value to a struct tm, adjusted to UTC
<u>localtime() and _localtime()</u>	Convert a time_t value to a struct tm
<u>mktime()</u>	Convert a struct tm to a time_t value
<u>strdate()</u>	Convert the current date to a string
<u>strftime()</u>	Format time and date string
<u>strtime()</u>	Convert current time of day to a string
<u>time()</u>	Get the current time and date
<u>tzset() and _tzset()</u>	Set local time-zone values
<u>utime() and _utime()</u>	Update a file's date/time stamp
<u>Environment Variable TZ</u>	