

MSL C Reference 9.0

Revised 20040427

metrowerks

Metrowerks and the Metrowerks logo are registered trademarks of Metrowerks Corporation in the United States and/or other countries. CodeWarrior is a trademark or registered trademark of Metrowerks Corporation in the United States and/or other countries. All other trade names and trademarks are the property of their respective owners.

Copyright © 2004 Metrowerks Corporation. ALL RIGHTS RESERVED.

No portion of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, without prior written permission from Metrowerks. Use of this document and related materials is governed by the license agreement that accompanied the product to which this manual pertains. This document may be printed for non-commercial personal use only in accordance with the aforementioned license agreement. If you do not have a copy of the license agreement, contact your Metrowerks representative or call 1-800-377-5416 (if outside the U.S., call +1-512-996-5300).

Metrowerks reserves the right to make changes to any product described or referred to in this document without further notice. Metrowerks makes no warranty, representation or guarantee regarding the merchantability or fitness of its products for any particular purpose, nor does Metrowerks assume any liability arising out of the application or use of any product described herein and specifically disclaims any and all liability. **Metrowerks software is not authorized for and has not been designed, tested, manufactured, or intended for use in developing applications where the failure, malfunction, or any inaccuracy of the application carries a risk of death, serious bodily injury, or damage to tangible property, including, but not limited to, use in factory control systems, medical devices or facilities, nuclear facilities, aircraft navigation or communication, emergency systems, or other applications with a similar degree of potential hazard.**

How to Contact Metrowerks

Corporate Headquarters	Metrowerks Corporation 7700 West Parmer Lane Austin, TX 78729 U.S.A.
World Wide Web	http://www.metrowerks.com
Sales	United States Voice: 800-377-5416 United States Fax: 512-996-4910 International Voice: +1-512-996-5300 Email: sales@metrowerks.com
Technical Support	United States Voice: 800-377-5416 International Voice: +1-512-996-5300 Email: support@metrowerks.com

Table of Contents

1	Introduction	27
	Organization of Files	27
	ANSI C Standard	29
	The ANSI C Library and Apple Macintosh	30
	MSL Extras Library	30
	POSIX Functionality	31
	Console I/O and the Macintosh	31
	Console I/O and Windows	31
	Using Mac OS X and the Extras Library	31
	Compatibility	32
	Intrinsic Functions	32
2	MSL C and Multithreading	33
	Introduction to Multithreading	33
	Definitions	34
	Reentrancy Functions	36
3	alloca.h	39
	Overview of alloca.h	39
	alloca	39
4	assert.h	41
	Overview of assert.h	41
	assert	41
5	conio.h	43
	Overview of conio.h	43
	clrscr	44
	getch	44
	getche	45
	gotoxy	45
	initscr	46

Table of Contents

inp	46
inp ^d	47
inp ^w	47
kbhit	48
outp	49
outpd.	49
outpw	50
_textattr.	50
_textbackground	51
_textcolor	52
_wherex	52
_wherey	53

6 console.h 55

Overview of console.h	55
ccommand	56
clrscr.	57
getch.	58
InstallConsole	58
kbhit	59
ReadCharsFromConsole	59
RemoveConsole	60
__ttynname.	60
WriteCharsToConsole	61

7 crtl.h 63

Overview of crtl.h	63
Argc	63
Argv	64
__DllTerminate	64
environ	64
__HandleTable	65
__CRTStartup.	65

_RunInit	66
_SetupArgs	66
8 ctype.h	67
Overview of ctype.h	67
Character testing and case conversion	67
Character Sets Supported	68
isalnum	68
isalpha	71
isblank	72
iscntrl	72
isdigit	73
isgraph	74
islower	74
isprint	75
ispunct	76
isspace	76
isupper	77
isxdigit	78
tolower	78
toupper	79
9 direct.h	81
Overview of direct.h	81
_getdcwd	81
_getdiskfree	82
_getdrives	82
10 dirent.h	85
Overview of dirent.h	85
opendir	85
readdir	86
readdir_r	87
rewinddir	88

Table of Contents

closedir	88
11 div_t.h	91
Overview of div_t.h	91
div_t	91
ldiv_t	92
lldiv_t	92
12 errno.h	95
Overview of errno.h	95
errno	95
13 extras.h	99
Overview of extras.h	99
_chdrive	101
chsize	102
filelength	102
fileno.	103
FullPath	104
gcvt	105
_getdrive	105
GetHandle	106
_get_osfhandle	106
heapmin	107
itoa	107
itow	108
ltoa	109
_ltow.	109
makepath	110
_open_osfhandle	111
putenv	111
_searchenv	112
splitpath	113
strcasecmp	114

strcmpl	114
strdate	115
strup	116
stricmpl	116
stricoll	117
strlwr.	118
strncasecmp	118
strncmpi	119
strncoll	120
strnicmp	120
strnicoll.	121
strnset	122
strrev	123
strset.	123
strspnp	124
strupr	125
tell.	125
ultoa.	127
_ultow	127
wcsdup	128
wcsicmp	129
wcsicoll	129
weslwr	130
wcsncoll	131
wesnicoll	131
wcsnicmp	132
wcsnset.	133
wcsrev	134
wcsset	134
wcsspnp	135
wcsupr	135
wstrrev	136
wtoi	136

Table of Contents

14 fcntl.h	139
Overview of fcntl.h	139
fcntl.h and UNIX Compatibility	139
creat, _wcreate	140
fcntl	141
open, _wopen	143
15 fenv.h	149
Overview of fenv.h	149
Data Types	149
fenv_t	149
fexcept_t	149
Macros	150
Floating-Point Exception Flags	150
Rounding Directions	150
Environment	151
Pragmas	151
FENV_ACC	151
Floating-point exceptions	151
feclearexcept	152
fegetexceptflag	153
feraiseexcept	154
fesetexceptflag	155
fetestexcept	156
Rounding	157
fegetround	158
fesetround	159
Environment	160
fegetenv	160
feholdexcept	161
fesetenv	162
feupdateenv	163

16 float.h	165
Overview of float.h	165
Floating point number characteristics	165
17 FSp_fopen.h	167
Overview of FSp_fopen.h	167
FSp_fopen	167
FSRef_fopen	168
FSRefParentAndFilename_fopen.	169
18 inttypes.h	171
Overview of inttypes.h	171
Greatest-Width Integer Types	171
imaxdiv_t	172
Greatest-Width FormatSpecifier Macros.	172
Greatest-Width Integer Functions.	174
imaxabs	174
imaxdiv	175
strtoimax	176
strtoumax.	177
wcstoimax	178
wcstoumax	179
19 io.h	181
Overview of io.h	181
_finddata_t	181
_findclose	182
_findfirst	183
_findnext	183
_setmode	184
20 iso646.h	187
Overview of iso646.h	187

Table of Contents

21 limits.h	189
Overview of limits.h	189
Integral type limits	189
22 locale.h	191
Overview of locale.h	191
Locale specification	191
localeconv	192
setlocale	193
23 malloc.h	195
Overview of malloc.h	195
alloca	195
Non Standard <malloc.h> Functions	196
24 math.h	197
Overview of math.h	197
Floating point mathematics	199
NaN Not a Number	199
Quiet NaN	200
Signaling NaN	200
Floating point error testing.	200
Inlined Intrinsics Option	201
Floating Point Classification Macros	201
Enumerated Constants	201
fpclassify	202
isfinite	203
isnan	203
isnormal	204
signbit	204
Floating Point Math Facilities	205
acos	205
acosf	206
acosl	206

asin	206
asinf	207
asinl	207
atan	208
atanf	208
atanl	209
atan2	209
atan2f	210
atan2l	210
ceil	211
ceilf	212
ceill	212
cos	213
cosf	214
cosl	214
cosh	214
coshf	215
coshl	215
exp	215
expf	217
expl	217
fabs	217
fabsf	218
fabsl	218
floor	218
floorf	219
floorl	220
fmod	220
fmodf	221
fmodl	221
frexp	222
frexpf	223
freexpl	223

Table of Contents

isgreater	223
isgreaterless	224
isless	224
islessequal	225
isunordered	225
ldexp	226
ldexpf	227
ldexpl	227
log	228
logf	229
logl	229
log10	230
log10f	230
log10l	231
modf	231
modff	232
modfl	232
pow	233
powf	234
powl	234
sin	235
sinf	236
sinl	236
sinh	237
sinhf	238
sinhl	238
sqrt	238
sqrtf	239
sqrtl	240
tan	240
tanf	241
tanl	241
tanh	241

tanhf	242
tanhl	243
HUGE_VAL	243
C99 Implementations	243
acosh	243
asinh	244
atanh	245
cbrt	246
copysign	247
erf	248
erfc	249
exp2	250
expm1	251
fdim	252
fma	253
fmax	254
fmin	255
gamma	256
hypot	257
ilogb	258
lgamma	259
log1p	260
log2	261
logb	262
modf	263
nan	264
nearbyint	264
nextafter	265
remainder	266
remquo	267
rint	268
rinttol	269
round	270

Table of Contents

roundtol	271
scalb	271
trunc	272
25 Process.h	275
Overview of Process.h	275
_beginthread	275
_beginthreadex	276
_endthread	277
_endthreadex	278
26 setjmp.h	279
Overview of setjmp.h	279
Non-local jumps and exception handling.	279
longjmp	280
setjmp	281
27 signal.h	285
Overview of signal.h	285
Signal handling.	285
signal	287
raise	290
28 SIOUX.h	291
Overview of SIOUX	291
Using SIOUX	291
SIOUX for Macintosh	292
Creating a Project with SIOUX	293
Customizing SIOUX	295
path2fss	301
SIOUXHandleOneEvent	302
SIOUXSetTitle	303

29 stat.h	305
Overview of stat.h	305
Stat Structure and Definitions	306
chmod	308
fstat	309
mkdir	311
stat	312
umask	315
30 stdarg.h	317
Overview of stdarg.h	317
Variable arguments for functions	317
va_arg	318
va_copy	318
va_end	319
va_start	320
31stdbool.h	323
Overview ofstdbool.h	323
32 stddef.h	325
Overview of stddef.h	325
NULL	325
offsetof	325
ptrdiff_t	326
size_t	326
wchar_t	326
33 stdint.h	329
Overview of stdint.h	329
Integer types	329
Limits of specified-width integer types	331
Macros for integer constants	335
Macros for greatest-width integer constants	335

Table of Contents

34 stdio.h	337
Overview of stdio.h	337
Standard input/output	339
Streams.	339
File position indicator	341
End-of-file and errors	341
Wide Character and Byte Character Stream Orientation	342
Stream Orientation and Standard Input/Output	343
clearerr	343
fclose	346
fdopen	348
feof	349
ferror.	352
fflush.	354
fgetc	356
fgetpos	358
fgets	360
_fileno	361
fopen.	361
fprintf	365
fputc	371
fputs	373
fread	375
freopen	378
fscanf	379
fseek	387
fsetpos	390
ftell	391
fwide.	392
fwrite	394
getc	395
getchar	397
gets	398

perror	400
printf	402
putc	410
putchar	412
puts	413
remove	414
rename	416
rewind	417
scanf	420
setbuf	425
setvbuf	428
snprintf	430
sprintf	431
sscanf	432
tmpfile	434
tmpnam.	435
ungetc	437
vfprintf	440
vfscanf	442
vprintf	445
vsnprintf	447
vsprintf	449
vsscanf	451
_wfopen	453
_wfreopen.	453
_wremove.	454
_wrename.	455
_wtmpnam	455
35 stdlib.h	457
Overview of stdlib.h	457
String Conversion Functions	457
Pseudo-random Number Generation Functions	458

Table of Contents

Memory Management Functions	458
Environment Communication Functions	458
Searching And Sorting Functions	459
Multibyte Conversion Functions	459
Integer Arithmetic Functions	459
abort	459
abs	461
atexit	462
atof	465
atoi	466
atol	467
atoll	468
bsearch	469
calloc	473
div	476
exit	477
_Exit	479
free	479
getenv	480
labs	481
ldiv	482
llabs	483
lldiv	483
malloc	484
mblen	485
mbstowcs	486
mbtowc	487
_putenv	488
qsort	488
rand	489
rand_r	491
realloc	491
srand	492

strtod	493
strtof	496
strtol	497
strtold	500
strtoll	501
strtoul	502
strtoull	503
system	505
vec_calloc	505
vec_free	506
vec_malloc	507
vec_realloc	508
wcstombs	508
wctomb	509
Non Standard <stdlib.h> Functions	510

36 string.h	511
Overview of string.h	511
memchr	512
memcmp	515
memcpy	516
memmove	517
memset	518
strcat	518
strchr	519
strcmp	520
strcoll	522
strcpy	524
strespn	525
strerror	527
strerror_r	528
strlen	528
strncat	529

Table of Contents

strncmp	531
strncpy	532
strpbrk	534
strrchr	535
strspn	536
strstr	537
strtok.	539
strxfrm	540
Non Standard <string.h> Functions	542
37 tgmath.h	545
Overview of tgmath.h	545
38 time.h	547
Overview of time.h	547
Date and time	548
Type clock_t.	548
Type time_t	548
struct tm	549
tzname	550
asctime	550
asctime_r	552
clock.	552
ctime.	555
ctime_r	556
difftime.	556
gmtime	558
gmtime_r	559
localtime	560
localtime_r	561
mktime	561
strftime	562
time	568

tzset	569
Non Standard <time.h> Functions	570
39 timeb.h	571
Overview of timeb.h	571
struct timeb	571
ftime	572
40 unistd.h	575
Overview of unistd.h	575
unistd.h and UNIX compatibility	576
access	576
chdir	577
close	579
cuserid	583
cwait	584
dup	584
dup2	585
exec functions	586
getcwd	588
getlogin	589
getpid	590
isatty	591
lseek	594
read	595
rmdir	596
sleep	598
spawn functions	600
ttyname	601
unlink	602
write	604
41 unix.h	607
Overview of unix.h	607

Table of Contents

UNIX Compatibility	607
_fcreator	607
_ftype	608
42 utime.h	611
Overview of utime.h	611
utime.h and UNIX Compatibility	611
utime	612
utimes	615
43 utsname.h	619
Overview of utsname.h	619
utsname.h and UNIX Compatibility	619
uname	620
44 wchar.h	623
Overview of wchar.h	623
Multibyte character functions	625
Wide Character and Byte Character Stream Orientation	625
Stream Orientation and Standard Input/Output	627
Definitions	627
btowc	628
fgetwc	628
fgetws	629
fputwc	630
fputws	630
fwprintf	631
fscanf	632
getwc	633
getwchar	634
mbrlen	634
mbrtowc	635
mbsinit	637
mbsrtowcs	637

putwc	638
putwchar	639
swprintf	640
swscanf.	641
vfwscanf	641
vswscanf	642
vwscanf	643
vfwprintf	644
vswprintf	646
vwprintf	647
watof.	648
wcrtomb	648
wescat	649
wcschr	650
wcscmp.	650
wcscoll	651
wcsespn	652
wescpy	652
wcsftime	653
wcslen	654
wcsncat.	654
wcsncmp	655
wcsncpy	656
wcspbrk	657
wesrchr.	657
wcsrtombs	658
wcsspn	659
wcsstr	660
wcstod	660
wcstof	661
wcstok	663
wcstol	664
wcstold	665

Table of Contents

wcstoll	666
wcstoul	667
wcstoull	669
wcsxfrm	670
wctime	671
wctob	671
wmemchr	672
wmemcmp	673
wmemcpy	674
wmemmove	674
wmemset	675
wprintf	676
wscanf	679
Non Standard <wchar.h> Functions	682

45 wctype.h	683
Overview of wctype.h	683
Types	684
iswalnum	684
iswalpha	685
iswblank	685
iswcntrl	686
iswdigit	686
iswgraph	687
iswlower	688
iswprint	688
iswpunct	689
iswspace	689
iswupper	690
iswxdigit	691
towctrans	691
towlower	692
towupper	692

wctrans	693
46 WinSIOUX.h	695
Overview of WinSIOUX	695
Using WinSIOUX	695
WinSIOUX for Windows	696
Creating a Project with WinSIOUX.	696
WinSIOUX.rc	697
Customizing WinSIOUX	697
clrscr	698
47 MSL Flags	701
Overview of the MSL Switches, Flags and Defines.	701
__ANSI_OVERLOAD__.	702
__MSL_C_LOCALE_ONLY	702
__MSL_IMP_EXP.	702
__MSL_INTEGRAL_MATH	703
__MSL_MALLOC_0 RETURNS_NONNULL	703
__MSL_NEEDS_EXTRAS	703
__MSL_OS_DIRECT_MALLOC	704
__MSL_CLASSIC_MALLOC	704
__MSL_USE_NEW_FILE_APIS	704
__MSL_USE_OLD_FILE_APIS	705
__MSL_POSIX	705
__MSL_STRERROR_KNOWS_ERROR_NAMES	705
__SET_ERRNO__	706
Index	707

Table of Contents

Introduction

This reference contains a description of the ANSI library and extended libraries bundled with Metrowerks C.

Organization of Files

The C headers files are organized alphabetically. Items within a header file are also listed in alphabetical order. Whenever possible, sample code has been included to demonstrate the use of each function.

The [“Introduction to Multithreading” on page 33](#) covers multithreading and thread-safeness of the Metrowerks Standard C Library functions.

The [“Overview of alloca.h” on page 39](#) covers the non-ANSI `alloca()` function for dynamic allocation from the stack.

The [“Overview of assert.h” on page 41](#) covers the ANSI C exception handling macro `assert()`.

The [“Overview of conio.h” on page 43](#) covers the Windows console input and output routines.

The [“Overview of console.h” on page 55](#) covers Macintosh console routines.

The [“Overview of crt.h” on page 63](#) covers Win32 console routines.

The [“Overview of ctype.h” on page 67](#) covers the ANSI character facilities.

The [“Overview of direct.h” on page 81](#) covers Win32x86 directory facilities.

The [“Overview of dirent.h” on page 85](#) covers various POSIX directory functions.

The [“Overview of div_t.h” on page 91](#) covers two arrays for math routines.

The [“Overview of errno.h” on page 95](#) covers ANSI global error variables.

The [“Overview of extras.h” on page 99](#) covers additional non standard functions included with the MSL library.

The [“Overview of fcntl.h” on page 139](#) covers non-ANSI control of files.

Introduction

Organization of Files

The “[Overview of fenv.h](#)” on page 149, covers floating-point environment facilities.

The “[Overview of float.h](#)” on page 165 covers ANSI floating point type limits.

The “[Overview of FSp_fopen.h](#)” on page 167, contains Macintosh file opening routines.

The “[Overview of inttypes.h](#)” on page 171 contains greatest-width integer routines and macros.

The “[Overview of io.h](#)” on page 181, contains common Windows stream input and output routines.

The “[Overview of iso646.h](#)” on page 187 contains operator symbol alternative words.

The “[Overview of limits.h](#)” on page 189 covers ANSI integral type limits.

The “[Overview of locale.h](#)” on page 191 covers ANSI character sets, numeric and monetary formats.

The “[Overview of malloc.h](#)” on page 195, covers the alloca function for Windows.

The “[Overview of math.h](#)” on page 197 covers ANSI floating point math facilities.

The “[Overview of Process.h](#)” on page 275, covers Windows thread process routines.

The “[Overview of setjmp.h](#)” on page 279 covers ANSI means used for saving and restoring a processor state.

The “[Overview of signal.h](#)” on page 285 covers ANSI software interrupt specifications.

The “[Overview of SIOUX](#)” on page 291 covers Metrowerks SIOUX console emulation for Macintosh.

The “[Overview of stat.h](#)” on page 305 covers non-ANSI file statistics and facilities.

The “[Overview of stdarg.h](#)” on page 317 covers ANSI custom variable argument facilities.

The “[Overview of stddef.h](#)” on page 325 covers the ANSI Standard Definitions.

The “[Overview of stdint.h](#)” on page 329 covers the latest integer types macros.

The “[Overview of stdio.h](#)” on page 337 covers ANSI standard input and output routines.

The “[Overview of stdlib.h](#)” on page 457 covers common ANSI library facilities.

The “[Overview of string.h](#)” on page 511 covers ANSI null terminated character array facilities.

The “[Overview of tgmath.h](#)” on page 545 lists type-generic math macros.

The “[Overview of time.h” on page 547](#) covers ANSI clock, date and time conversion and formatting facilities.

The “[Overview of timeb.h” on page 571](#) allows for programmer allocation of a buffer to store time information.

The “[Overview of unistd.h” on page 575](#) covers many of the common non-ANSI facilities.

The “[Overview of unix.h” on page 607](#) covers some Metrowerks non-ANSI facilities.

The “[Overview of utime.h” on page 611](#) covers non-ANSI file access time facilities.

The “[Overview of utsname.h” on page 619](#) covers the non-ANSI equipment naming facilities.

The “[Overview of wchar.h” on page 623](#) covers the wide character set for single and array facilities.

The “[Overview of wctype.h” on page 683](#) covers the wide character set type comparison facilities.

The “[Overview of WinSIOUX” on page 695](#) covers Metrowerks SIOUX console emulation for Windows.

The “[Overview of the MSL Switches, Flags and Defines” on page 701](#) covers the various switches, flags and defines used to customize the MSL C library.

ANSI C Standard

The ANSI C Standard Library included with Metrowerks CodeWarrior follows the specifications in the ANSI: Programming Language C / X3.159.1989 document together with extensions according to ISO/IEC 9899:1999 (known for some time as “C99”). The functions, variables and macros available in this library can be used transparently by both C and C++ programs.

Metrowerks Standard Library implements internal macros, _MSL_C99, that separate those parts of the C library that were added to the first version of the ANSI Standard for the C programming language (ANSI/ISO 9899-1990) by the second version (ISO/IEC 9899-1999(E)). If _MSL_C99 is defined in a prefix file to have the value 0 before building the MSL C library, only those parts of the library that were defined in ANSI/ISO 9899-1990 are compiled, yielding a smaller library. If _MSL_C99 is defined to have a non-zero value before building the library the full MSL C library is compiled in accord with ISO/IEC 9899-1999(E).

The ANSI C Library and Apple Macintosh

Some functions in the ANSI C Library are not fully operational on the Macintosh environment because they are meant to be used in a character-based user interface instead of the Macintosh computer's graphical user interface. While these functions are available, they may not work as you expect them to. Such inconsistencies between the ANSI C Standard and the Metrowerks implementation are noted in a function's description.

Except where noted, ANSI C Library functions use C character strings, not Pascal character strings.

MSL Extras Library

The MSL Extras Library contains functions macros and types that are not included in the ANSI/ISO C Standard as well as POSIX functions. These are included for Windows and UNIX compatibility. See the description of “[MSL NEEDS EXTRAS](#)” on page 703, for access of these functions when including a C standard header.

The functions, procedures and types in the headers listed in “[MSL Extras Library Headers](#)” on page 30 are included in the MSL Extras Library.

Table 1.1 MSL Extras Library Headers

dirent.h	extras.h	fcntl.h	stat.h
unistd.h	unix.h	utime.h	utsname.h
Mac OS Only	Console.h		
Windows Only	conio.h	direct.h	io.h

Windows Only

On Windows the MSL extras functions are enabled using the same name with a leading underscore.

POSIX Functionality

The Metrowerks Standard Libraries includes some but not all `POSIX` functions and types.

Console I/O and the Macintosh

The ANSI Standard Library assumes interactive console I/O (the `stdin`, `stderr`, and `stdout` streams) is always open. Many of the functions in this library were originally designed to be used on a character-oriented user interface, not the graphical user interface of a Macintosh computer. These header files contain functions that help you run character-oriented programs on a Macintosh:

- `console.h` declares `ccommand()`, which displays a dialog that lets you enter command-line arguments
- `SIOUX.h` is part of the SIOUX package, which creates a window that's much like a dumb terminal or TTY. Your program uses that window whenever your program refers to `stdin`, `stdout`, `stderr`, `cin`, `cout`, or `cerr`.

Console I/O and Windows

The ANSI Standard Library assumes interactive console I/O (the `stdin`, `stderr`, and `stdout` streams) is always open. This commandline interface is provided by the Windows console applications. You may want to check the headers `io.h`, `crtl.h` and `process.h` for specific Windows console routines. In addition, `WinSIOUX.h` and the `WinSIOUX` libraries provide a the Graphical User Interface consisting of a window that is much like a dumb terminal or TTY but with scrolling and cut and paste facilities.

Using Mac OS X and the Extras Library

On OS X, the functions which would previously have come from `MSL_Extras` are available from the `System.framework` using headers from the {OS X

Volume } :usr:include:access path. Therefore Mach-O on Mac OS X requires access path settings so as to not use the MSL Extras library.

Do not use an access path to the top level {Compiler}:MSL: directory as that will bring in files from the new MSL_Extras.

Compatibility

Parts of the Metrowerks Standard Library including both the ISO Standard C library and POSIX conforming procedures and definitions, as well as MSL library extensions, may not be implemented on all platforms. Furthermore, information about your target may not appear in this version of the printed documentation. You should consult the electronic documentation or release notes for your product to determine whether a particular function is compatible with your target platform.

Intrinsic Functions

Intrinsic functions generate in-line assembly code to perform the library routine. Generally these exist to allow direct access to machine functions which are not easily expressed directly in C. In some cases these map to single assembler instructions.

Some examples of intrinsics are

```
long __labs(long);  
double __fabs(double);
```

and

```
void *__memcpy(void *, const void *, size_t);
```

where `__memcpy()` provides access to the block move in the code generator to do the block move inline.

MSL C and Multithreading

This reference contains a description of multithreading and thread safety in the Metrowerks C library.

Introduction to Multithreading

In programming, the term thread is generally used to refer to the smallest amount of processor context state necessary to encapsulate a computation. Practically speaking, a thread consists of a register set, and a stack together with the address space where data is stored. Some parts of this address space are private to the thread while other parts may be shared with other threads. Variables that belong to the storage class `auto` and that are instantiated by the thread are private to the thread and cannot be accessed by other threads. This is in contrast to variables that belong to the storage class `static`, which may be accessed by other threads in the same process. All threads also have access to the standard files, `stdin`, `stdout`, and `stderr`. In addition, a multithreading implementation may provide for data with the same kind of lifetime as data in the `static` storage class but where access is restricted to the thread that owns it. Such data is known as `thread-local` data.

Most current operating systems are said to be multithreaded because they allow the creation of additional threads within a process beyond the one that begins the execution of the process. Thus, in a multithreaded operating system, a process may consist of more than one thread, all of them executing simultaneously.

Of course, unless there is more than one processor, the threads are not really executing simultaneously; the operating system is giving the impression of simultaneity by multiplexing between the threads. The operating system determines which thread is to have control of the processor at any particular time. There are two models for operating a multithreaded process, the cooperative model and the preemptive model. In the cooperative model, the threads signal their willingness to relinquish their control of the processor through a system call and the operating system then decides which thread will gain control. Thus, in this model, the execution of a thread can only be interrupted at points that are convenient to the algorithm. In the preemptive model, the operating system will switch between the threads at arbitrary and unpredictable times and points in the code being executed. Such a thread switch may occur between any

two machine instructions and not coincide with a boundary between two C statements; it may even be part-way through the evaluation of an expression. One important result of this is that, since switching between threads happens unpredictably, if more than one thread is changing the value of a shared variable, the results of an execution are likely to differ from one run to another. This lack of repeatability makes debugging and validation difficult. In what follows, we shall be assuming a preemptive model of multithreading.

Multithreading calls for mechanisms to protect against such uncertainty. Various methods exist for protecting segments of code from being executed by two threads at the same time. A program that is suitably protected against errors in the presence of multithreading is said to be thread-safe.

Definitions

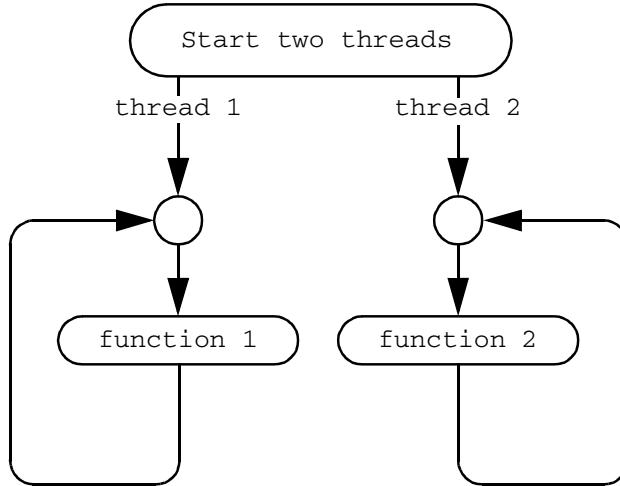
Essentially, there are no standards for implementing multithreading. In particular, neither the C99 Standard nor the POSIX Standard makes reference to threads. For the MSL C library, we define thread-safety as:

- An MSL C Library function will be said to be “thread-safe” if a single invocation of the function can be viewed as a single uninterruptable (i.e. atomic) operation. That is, it is possible to have two simultaneously executing threads in a single process both using the function without danger of mutual interference.

For most functions in the library, the meaning of this is fairly clear. Some functions, such as `rand()` or `strtok()` are defined in C99 to maintain internal state variables and would appear, by definition, to be not thread-safe. However, in MSL on Windows, functions make use of thread-local data to ensure their thread-safety.

A model for a test program to demonstrate the mutual thread safety of two library functions

Figure 2.1 Thread Safety Test



where:

- Function 1 and function 2 can be the same or different library functions.
- Each loop has to iterate many times.

For these two functions to be said to be mutually threadsafe, the results produced by thread 1 must be exactly the same as they would be if thread 2 did not exist.

Thread safety problems only arise if the two threads are sharing data, for example if function 1 and function 2 are both writing to the same file or if they both use a function that maintains its own internal state without protecting it in thread-local data, as is done in `strtok()`.

The following MSL C library functions have special precautions to make them threadsafe:

Listing 2.1 Functions with special threat precautions

asctime	atexit	calloc	ctime	exit	fgetc
fgetpos	fgets	fgetwc	fgetws	fopen	fprintf
fputc	fputs	fputwc	fputws	fread	free
fscanf	fseek	fsetpos	ftell	fwprintf	fwrite
fwscanf	getc	getchar	gets	getwc	getwchar
gmtime	localeconv	localtime	malloc	printf	putc
putchar	puts	putwc	putwchar	raise	rand

Listing 2.1 Functions with special threat precautions

realloc	scanf	setbuf	setlocale	setvbuf	signal
srand	strtok	tmpfile	tmpnam	ungetc	ungetwc
vfprintf	vfscanf	vfwprintf	vfwscanf	vprintf	vscanf
vwprintf	vwscanf	wcrtomb	wcsrtombs	wctomb	wprintf
wscanf					

The remaining MSL C functions are intrinsically thread-safe.

Reentrancy Functions

In the most recent versions of MSL C, great pains have been taken to make sure that every function works properly in a multithreaded environment. The ANSI C standard makes no provisions at all for thread safety. Unlike some other library vendors, MSL C takes the standpoint that all functions should be thread safe, providing that the `_MSL_THREADS` macro is defined to 1.

When the `_MSL_THREADS` macro is 0, many of the MSL C library functions lose their thread safe attributes. However, it may be useful to leave the `_MSL_THREADS` macro as 0 even on a multithreaded system for the reasons of speed. The library functions will be faster if they do not have to wait for thread synchronization. Since many programs are written using only a single thread, it is often advantageous to provide an efficient single threaded library.

GCC and other library vendors provide an assortment of helper functions, all with a `_r` suffix, to indicate they are naturally thread safe.

The following is a list of enhanced header files and what new "`_r`" functions they implement:

- dirent.h
 - readdir_r
- stdlib.h
 - rand_r
- string.h
 - strerror_r
- time.h
 - asctime_r

- ctime_r
- gmtime_r
- localtime_r

alloca.h

This header defines one function, [alloca](#), which lets you allocate memory quickly using the stack.

Overview of alloca.h

The alloca.h header file consists of

- [“alloca” on page 39](#) that allocates memory from the stack

alloca

Allocates memory quickly on the stack.

```
#include <alloca.h>
void *alloca(size_t nbytes);
```

nbytes	size_t	number of bytes of allocation
--------	--------	-------------------------------

Remarks

This function returns a pointer to a block of memory that is nbytes long. The block is on the function’s stack. This function works quickly since it decrements the current stack pointer. When your function exits, it automatically releases the storage.

NOTE The AltiVec version of `alloca()` allocates memory on a 16 byte alignment.

If you use `alloca()` to allocate a lot of storage, be sure to increase the Stack Size for your project in the Project preferences panel.

If it is successful, `alloca()` returns a pointer to a block of memory. If it encounters an error, `alloca()` returns `NULL`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“calloc” on page 473](#), [“free” on page 479](#)

[“malloc” on page 484](#), [“realloc” on page 491\)](#)

assert.h

The assert.h header file provides a debugging macro, [assert](#), that outputs a diagnostic message and stops the program if a test fails.

Overview of assert.h

The assert.h header file provides a debugging macro, [“assert” on page 41](#), that outputs a diagnostic message and stops the program if a test fails.

assert

Abort a program if a test is false.

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

expression	int	A boolean expression being evaluated
------------	-----	--------------------------------------

Remarks

If expression is false the assert() macro outputs a diagnostic message to stderr and calls abort(). The diagnostic message has the form

```
file: line test -- assertion failed  
abort -- terminating
```

where file is the source file, line is the line number, and test is the failed expression.

To turn off the assert() macros, place a #define NDEBUG (no debugging) directive before the #include <assert.h> directive.

assert.h

Overview of assert.h

This macro may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“abort” on page 459](#)

Listing 4.1 Example of assert() usage.

```
#undef NDEBUG
/* Make sure that assert() is enabled */
#include <assert.h>
#include <stdio.h>

int main(void)
{
    int x = 100, y = 5;
    printf("assert test.\n");

    /*This assert will output a message and abort the program */
    assert(x > 1000);
    printf("This will not execute if NDEBUG is undefined\n");
    return 0;
}
```

Output:
assert test.
foo.c:12 x > 1000 -- assertion failed
abort -- terminating

conio.h

The `conio.h` header file consists of various runtime declarations that pertain to the Win32 x86 targets for console input and output.

Overview of conio.h

This header defines the following facilities.

- [“`clrscr`” on page 44](#) clears the console window
- [“`getch`” on page 44](#) reads a char from the input screen
- [“`getche`” on page 45](#) reads a char and echoes to the screen
- [“`gotoxy`” on page 45](#) places the cursor on a console window
- [“`initscr`” on page 46](#) sets up a console in a GUI application
- [“`inp`” on page 46](#) reads a byte from an input port
- [“`inpD`” on page 47](#) reads a double word from an input port
- [“`inpW`” on page 47](#) reads a word from an input port
- [“`kbhit`” on page 48](#) returns true if a key is pressed
- [“`outp`” on page 49](#) outputs a byte to a port
- [“`outpD`” on page 49](#) outputs a double word to a port
- [“`outpW`” on page 50](#) outputs a word to a port
- [“`textattr`” on page 50](#) sets the text attributes
- [“`textbackground`” on page 51](#) sets the text background color
- [“`textcolor`” on page 52](#) sets the text color
- [“`wherex`” on page 52](#) returns the horizontal coordinate
- [“`wherey`” on page 53](#) returns the vertical coordinate

_clrscr

Clears the standard output screen.

```
#include <conio.h>
void _clrscr(void);
```

Remarks

This facility has no parameters.

No value is returned in this implementation.

This function is Windows only when declared from this header.

getch

This function reads a single char from the standard input device and does not echo it to the output without the need to press the enter key.

```
#include <conio.h>
int getch(void);
int _getch(void);
```

Remarks

This facility has no parameters.

The function `getch` returns the char read.

This function is Windows only when declared from this header.

See Also

[“getc” on page 395](#)

[“getchar” on page 397](#)

getche

This function reads a single char from the standard input device and echoes it to the output without the need of pressing the enter key.

```
#include <conio.h>
int getche(void);
int _getche(void);
```

Remarks

This facility has no parameters.

The function `getche` returns the char read.

This function is Windows only when declared from this header.

See Also

[“getc” on page 395](#)

[“getchar” on page 397](#)

_gotoxy

Moves the cursor to the horizontal and vertical coordinates in a standard output device.

```
#include <conio.h>
void _gotoxy(int x, int y);
```

x	int	vertical screen coordinate
y	int	horizontal screen coordinate

Remarks

No value is returned in this implementation.

This function is Windows only when declared from this header.

See Also

[“ wherex” on page 52](#)

[“ wherey” on page 53](#)

_initscr

Sets up standard 80x25 console with no scrolling region.

```
#include <conio.h>
void _initscr(void);
```

Remarks

This facility has no parameters.

There is no need to call `_initscr()` unless you have a GUI application (subsystem Windows GUI), where no console is available at runtime.

No value is returned in this implementation.

This function is Windows only when declared from this header.

inp

Reads a byte from the specified port

```
#include <conio.h>
unsigned char inp (unsigned short port);
unsigned char _inp (unsigned short port);
```

port	unsigned short	a port specified by number
------	----------------	----------------------------

Remarks

The value as a byte read is returned.

This function is Windows only when declared from this header.

See Also

[“inpd” on page 47](#)

[“inpw” on page 47](#)

inpd

Reads a double word from the specified port

```
#include <conio.h>
unsigned long inpd (unsigned short port);
unsigned long _inpd (unsigned short port);
```

port	unsigned short	a port specified by number
------	----------------	----------------------------

Remarks

The value as a long read is returned.

This function is Windows only when declared from this header.

See Also

[“inp” on page 46](#)

[“inpw” on page 47](#)

inpw

Reads a word from the specified port

```
#include <conio.h>
unsigned short inpw (unsigned short port);
unsigned short _inpw (unsigned short port);
```

port	unsigned short	a port specified by number
------	----------------	----------------------------

Remarks

The value as a word read is returned.

This function is Windows only when declared from this header.

See Also

[“inp” on page 46](#)

[“inpd” on page 47](#)

kbhit

This function is used in a loop to detect an immediate keyboard key press.

```
#include <conio.h>
int kbhit(void);
int _kbhit(void);
```

This facility has no parameters.

Remarks

The `kbhit` function has the side effect of discarding any non-keyboard input records in the console input queue

If the keyboard is pressed `kbhit()` returns a non zero value otherwise it return zero.

This function is Windows only when declared from this header.

See Also

[“getch” on page 44](#)

[“getche” on page 45](#)

outp

Outputs a byte to a specified port.

```
#include <conio.h>

void outp (unsigned short pt, unsigned char out);
void _outp (unsigned short pt, unsigned char )out;
```

port	unsigned short	a port specified by number
out	unsigned char	a byte value sent to the output

Remarks

No value is returned in this implementation.

This function is Windows only when declared from this header.

See Also

[“outpd” on page 49](#)

[“outpw” on page 50](#)

outpd

Sends a double word to the output port specified.

```
#include <conio.h>

void outpd (unsigned short pt, unsigned long out);
void _outpd (unsigned short pt, unsigned long out);
```

pt	unsigned short	a port specified by number
out	unsigned long	a double word value sent to the output

No value is returned in this implementation.

This function is Windows only when declared from this header.

See Also

[“outp” on page 49](#)

[“outpw” on page 50](#)

outpw

Sends a word to the output port specified.

```
#include <conio.h>

void outpw (unsigned short pt, unsigned short out);
void _outpw (unsigned short pt, unsigned short out);
```

pt	unsigned short	a port specified by number
out	unsigned long	a word value sent to the output

Remarks

No value is returned in this implementation.

This function is Windows only when declared from this header.

See Also

[“outp” on page 49](#)

[“outpd” on page 49](#)

_textattr

This function sets the attributes of the console text.

```
#include <conio.h>

void _textattr(int newattr);
```

newattr	int	the text attributes to be set
---------	-----	-------------------------------

Remarks

The function `_textattr` allows you to set both the foreground and background attributes with the variable `newattr`. The attributes available for this function are defined in the header file `wincon.h`.

No value is returned in this implementation.

This function is Windows only when declared from this header.

See Also

[“_textbackground” on page 51](#)

[“_textcolor” on page 52](#)

_textbackground

This function sets the console's background color.

```
#include <conio.h>
void _textbackground(int newcolor);
```

newcolor	int	the background color to be set
----------	-----	--------------------------------

Remarks

The attributes available for this function are defined in the header file `wincon.h`.

No value is returned in this implementation.

This function is Windows only when declared from this header.

See Also

[“_textattr” on page 50](#)

[“_textcolor” on page 52](#)

_textcolor

This function sets the console's text color.

```
#include <conio.h>
void _textcolor(int newcolor);
```

newcolor	int	the text color to be set
----------	-----	--------------------------

Remarks

The attributes available for this function are defined in the header file *wincon.h*.

No value is returned in this implementation.

This function is Windows only when declared from this header.

See Also

[“_textattr” on page 50](#)
[“_textbackground” on page 51](#)

_wherex

Determines the horizontal coordinate of the cursor in a console window.

```
#include <conio.h>
int _wherex(void);
```

Remarks

This facility has no parameters.

Returns the horizontal coordinate.

This function is Windows only when declared from this header.

See Also

[“_wherey” on page 53](#)

[“_gotoxy” on page 45](#)

_wherex

Determines the vertical coordinate of the cursor in a console window.

```
#include <conio.h>
int _wherex(void);
```

Remarks

This facility has no parameters.

Returns the vertical coordinate.

This function is Windows only when declared from this header.

See Also

[“_gotoxy” on page 45](#)

[“_wherex” on page 52](#)

conio.h

Overview of conio.h

console.h

This header file contains procedures and types, which help you port a program that was written for a command-line/console interface to the Macintosh operating system.

The console.h header file consists of various runtime declarations that pertain to the Classic and Carbon Macintosh interfaces.

Overview of console.h

This header file contains one function

- [“ccommand” on page 56](#), which helps you port a program that relies on command-line arguments.
- [“clrscr” on page 57](#), clears the SIOUX window and flushes the buffer.
- [“getch” on page 58](#) returns the keyboard character pressed when an ascii key is pressed
- [“InstallConsole” on page 58](#) installs the Console package.
- [“kbhit” on page 59](#) returns true if any keyboard key is pressed without retrieving the key
- [“ReadCharsFromConsole” on page 59](#) reads from the Console into a buffer.
- [“RemoveConsole” on page 60](#) removes the console package.
- [“ttynname” on page 60](#) Returns the name of the terminal associated with the file id. The unix.h function ttynname calls this function
- [“WriteCharsToConsole” on page 61](#) writes a stream of output to the Console window.

ccommand

Lets you enter command-line arguments for a SIOUX program.

```
#include <console.h>
int ccommand(char ***argv);
```

argv	char ***	The address of the second parameter of your command line
------	----------	--

Remarks

The function `ccommand()` must be the first code generated in your program. It must directly follow any variable declarations in the main function.

This function displays a dialog that lets you enter arguments and redirect standard input and output. Please refer to [“Overview of SIOUX” on page 291](#), for information on customizing SIOUX, or setting console options.

Only `stdin`, `stdout`, `cin` and `cout` are redirected. Standard error reporting methods `stderr`, `cerr` and `clog` are not redirected.

The maximum number of arguments that can be entered is determined by the value of `MAX_ARGS` defined in `ccommand.c` and is set to 25. Any arguments in excess of this number are ignored.

Enter the command-line arguments in the Argument field. Choose where your program directs standard input and output with the buttons below the field: the buttons on the left are for standard input and the buttons on the right are for standard output. If you choose Console, the program reads from or write to a SIOUX window. If you choose File, `ccommand()` displays a standard file dialog which lets you choose a file to read from or write to. After you choose a file, its name replaces the word *File*.

The function `ccommand()` returns an integer and takes one parameter which is a pointer to an array of strings. It fills the array with the arguments you entered in the dialog and returns the number of arguments you entered. As in UNIX or DOS, the first argument, the argument in element 0, is the name of the program. [“Example of `ccommand\(\)`” on page 57](#) has an example of command line usage

This function returns the number of arguments you entered.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

[“Customizing SIOUX” on page 295](#)

Listing 6.1 Example of ccommand()

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

int main(int argc, char *argv[])
{
    int i;

    argc = ccommand(&argv);

    for (i = 0; i < argc; i++)
        printf("%d. %s\n", i, argv[i]);
    return 0;
}
```

clrscr

Clears the console window and flushes the buffers;

```
#include <console.h>
void clrscr(void);
```

Remarks

This function is used to select all and clear the screen and buffer by calling SIOUXclrscr from SIOUX.h.

Macintosh only, This function may not be implemented on all Mac OS versions.

getch

Returns the keyboard character pressed when an ascii key is pressed

```
#include <console.h>
int getch(void);
```

Remarks

This function is used for console style menu selections for immediate actions.

Returns the keyboard character pressed when an ascii key is pressed.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

[“kbhit” on page 59](#)

InstallConsole

Installs the Console package.

```
#include <console.h>
extern short InstallConsole(short fd);
```

fd	short	A file descriptor for standard i/o
----	-------	------------------------------------

Remarks

Installs the Console package, this function will be called right before any read or write to one of the standard streams.

This function returns any error.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

[“Customizing SIOUX” on page 295](#)

[“RemoveConsole” on page 60](#)

kbhit

Returns true if any keyboard key is pressed.

```
#include <console.h>
int kbhit(void);
```

Remarks

Returns true if any keyboard key is pressed without retrieving the key used for stopping a loop by pressing any key

This function returns non zero when any keyboard key is pressed.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

[“getch” on page 58](#)

ReadCharsFromConsole

Reads from the Console into a buffer.

```
#include <console.h>
extern long ReadCharsFromConsole
(char *buffer, long n);
```

buffer	char *	A stream buffer
n	long	Number of char to read

Remarks

Reads from the Console into a buffer. This function is called by read.

Any errors encountered are returned.

Macintosh only, This function may not be implemented on all Mac OS versions.

console.h

Overview of console.h

See Also

[“WriteCharsToConsole” on page 61](#)

RemoveConsole

Removes the console package.

```
#include <console.h>
extern void RemoveConsole(void);
```

Remarks

Removes the console package. It is called after all other streams are closed and exit functions (installed by either atexit or __atexit) have been called.

Since there is no way to recover from an error, this function doesn't need to return any.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

[“Customizing SIOUX” on page 295](#)

[“InstallConsole” on page 58](#)

__ttyname

Returns the name of the terminal associated with the file id.

```
#include <console.h>
extern char *__ttyname(long fildes);
```

fildes	long	The file descriptor
--------	------	---------------------

Remarks

Returns the name of the terminal associated with the file id. The unix.h function ttynname calls this function (we need to map the int to a long for size of int variance).

Returns the name of the terminal associated with the file id.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

[“ttynname” on page 601](#)

WriteCharsToConsole

Writes a stream of output to the Console window.

```
#include <console.h>
extern long WriteCharsToConsole (char *buffer, long n);
```

buffer	char *	A stream buffer
n	long	Number of char to write

Remarks

Writes a stream of output to the Console window. This function is called by write.

Any errors encountere are returned.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

[“ReadCharsFromConsole” on page 59](#)

console.h

Overview of console.h

crtl.h

The `crtl.h` header file consist of various runtime declarations that pertain to the Win32 x86 targets.

Overview of `crtl.h`

This header defines the following facilities.

- [“`Argc`” on page 63](#), is the argument list count
- [“`Argv`” on page 64](#), the argument list variables
- [“`DllTerminate`” on page 64](#), shows when a DLL is running terminate code.
- [“`environ`” on page 64](#), is the environment pointers
- [“`HandleTable`” on page 65](#), is a structure allocated for each ed file handle
- [“`CRTStartup`” on page 65](#), initializes the C Runtime start-up routines.
- [“`RunInit`” on page 66](#), initializes the runtime, static classes and variables.
- [“`SetupArgs`” on page 66](#), sets up the command line arguments.

Argc

The argument count variable

```
#include <crtl.h>
extern int __argc;
```

Remarks

Used for command line argument count.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Argv

The argument command variables.

```
#include <crtl.h>
extern char **__argv;
```

Remarks

The command line arguments.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

_DlTerminate

A flag to determine when a DLL is running terminate code.

```
#include <crtl.h>
extern int _DlTerminate;
```

Remarks

This flag is set when a DLL is running terminate code.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

environ

The environment pointers

```
#include <crtl.h>
extern char * (*environ);
```

Remarks

This is a pointer to the environment.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

_HandleTable

FileStruct is a structure allocated for each file handle

```
#include <crtl.h>
typedef struct
{
    void *handle;
    char translate;
    char append;
} FileStruct;

extern FileStruct *_HandleTable[NUM_HANDLES];
extern int _HandPtr;
```

Remarks

The variable `_HandPtr` is a pointer to a table of handles.

The variable `NUM_HANDLES` lists the number of possible handles.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

_CRTStartup

The function `_CRTStartup` is the C Runtime start-up routine.

```
#include <crtl.h>
extern void _CRTStartup();
```

Remarks

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

_RunInit

The function `_RunInit` initializes the runtime, all static classes and variables.

```
#include <crtl.h>
extern void _RunInit();
```

Remarks

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

_SetupArgs

The function `_SetupArgs` sets up the command line arguments.

```
#include <crtl.h>
extern void _SetupArgs();
```

Remarks

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

ctype.h

The `ctype.h` header file supplies macros and functions for testing and manipulation of character type.

Overview of ctype.h

Character testing and case conversion

The `ctype.h` header file supplies macros for testing character type and for converting alphabetic characters to uppercase or lowercase. The `ctype.h` macros support ASCII characters (0x00 to 0x7F), and the EOF value. These macros are not defined for the Apple Macintosh Extended character set (0x80 to 0xFF).

The header `ctype.h` includes several function for testing of character types. The include:

- [“isalnum” on page 68](#) tests for alphabetical and numerical characters
- [“isalpha” on page 71](#) tests for alphabetical characters
- [“isblank” on page 72](#) tests for a space between words and is dependent upon the locale usage
- [“iscntrl” on page 72](#) tests for control characters
- [“isdigit” on page 73](#) tests for digit characters
- [“isgraph” on page 74](#) tests for graphical characters
- [“islower” on page 74](#) tests for lower case characters
- [“ispunct” on page 75](#) tests for punctuation characters
- [“isspace” on page 76](#) tests for white space characters
- [“isupper” on page 77](#) test for upper case characters
- [“isxdigit” on page 78](#) texts for hexadecimal characters
- [“tolower” on page 78](#) changes from uppercase to lowercase

-
- “[toupper](#)” on page 79 changes from lower case to uppercase

Character Sets Supported

Metrowerks Standard Library character tests the ASCII character set. Testing of extended character sets is undefined and may or may not work for any specific system. See “[Character Testing Functions](#)” on page 68 for return values.

Listing 8.1 Character Testing Functions

This function	Returns true if c is
isalnum(c)	Alphanumeric: [a-z], [A-Z], [0-9]
isalpha(c)	Alphabetic: [a-z], [A-Z].
isblank(c)	A blankspace between words based on locale
iscntrl(c)	The delete character (0x7F) or an ordinary control character from 0x00 to 0x1F.
isdigit(c)	A numeric character: [0-9].
isgraph(c)	A non-space printing character from the exclamation (0x21) to the tilde (0x7E).
islower(c)	A lowercase letter: [a-z].
isprint(c)	A printable character from space (0x20) to tilde (0x7E).
ispunct(c)	A punctuation character. A punctuation character is neither a control nor an alphanumeric character.
isspace(c)	A space, tab, return, new line, vertical tab, or form feed.
isupper(c)	An uppercase letter: [A-Z].
isxdigit(c)	A hexadecimal digit [0-9], [A-F], or [a-f].

isalnum

Determine character type.

```
#include <ctype.h>
int isalnum(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c. For example usage see [“Character testing functions example” on page 70](#).

In the “C” locale, isalpha returns true only for the characters for which isupper or islower is true.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“tolower” on page 78](#)

[“toupper” on page 79](#)

ctype.h

Overview of ctype.h

Listing 8.2 Character testing functions example

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    char *test = "Fb6# 9,";

    isalnum(test[0]) ?
        printf("%c is alpha numerical\n", test[0]) :
        printf("%c is not alpha numerical\n", test[0]);
    isalpha(test[0]) ?
        printf("%c is alphabetical\n", test[0]) :
        printf("%c is not alphabetical\n", test[0]);
    isblank(test[4]) ?
        printf("%c is a blank sapce\n", test[4]) :
        printf("%c is not a blank space\n", test[4]);
    iscntrl(test[0]) ?
        printf("%c is a control character\n", test[0]) :
        printf("%c is not a control character\n", test[0]);
    isdigit(test[2]) ?
        printf("%c is a digit\n", test[2]) :
        printf("%c is not a digit\n", test[2]);
    isgraph(test[0]) ?
        printf("%c is graphical \n", test[0]) :
        printf("%c is not graphical\n", test[0]);
    islower(test[1]) ?
        printf("%c is lower case \n", test[1]) :
        printf("%c is not lower case\n", test[1]);
    isprint(test[3]) ?
        printf("%c is printable\n", test[3]) :
        printf("%c is not printable\n", test[3]);
    ispunct(test[6]) ?
        printf("%c is a punctuation mark\n", test[6]) :
        printf("%c is not punctuation mark\n", test[6]);
    isspace(test[4]) ?
        printf("%c is a space\n", test[4]) :
        printf("%c is not a space\n", test[4]);
    isupper(test[0]) ?
        printf("%c is upper case \n", test[1]) :
        printf("%c is not upper case\n", test[1]);
    isxdigit(test[5]) ?
        printf("%c is a hex digit\n", test[5]) :
        printf("%c is not a hex digit\n", test[5]);
    return 0;
}
```

Output:

```
F is alpha numerical
F is alphabetical
    is a blank sapce
F is not a control character
6 is a digit
F is graphical
b is lower case
# is printable
, is a punctuation mark
    is a space
b is upper case
9 is a hex digit
```

isalpha

Determine character type.

```
#include <ctype.h>
int isalpha(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of *c*.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

isblank

Tests for a blank space or a word separator dependent upon the locale usage.

```
#include <ctype.h>
int isblank(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This function determines if a character is a blank space or tab or if the character is in a locale specific set of characters for which isspace is true and is used to separate words in text.

In the “C” locale, isblank returns true only for the space and tab characters.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isspace” on page 76](#)

iscntrl

Determine character type.

```
#include <ctype.h>
int iscntrl(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

isdigit

Determine character type.

```
#include <ctype.h>
int isdigit(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

isgraph

Determine character type.

```
#include <ctype.h>
int isgraph(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

islower

Determine character type.

```
#include <ctype.h>
int islower(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c.

In the “C” locale, `islower` returns true only for the lowercase characters.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

isprint

Determine character type.

```
#include <ctype.h>  
  
int isprint(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

ispunct

Determine character type.

```
#include <ctype.h>
int ispunct(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c.

In the “C” locale, `ispunct` returns true for every printing character for which neither `isspace` nor `isalnum` is true.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

isspace

Determine character type.

```
#include <ctype.h>
int isspace(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c.

In the “C” locale, `isspace` returns `true` only for the standard white-space characters.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

isupper

Determine character type.

```
#include <ctype.h>
int isupper(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of `c`.

In the “C” locale, `isupper` returns `true` only for the uppercase characters.

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

isxdigit

Determine hexadecimal type.

```
#include <ctype.h>
int isxdigit(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

This macro returns nonzero for true, zero for false, depending on the integer value of c. For example usage see [“Character testing functions example” on page 70](#)

[“Character Testing Functions” on page 68](#) describes what the character testing functions return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

For example usage see [“Character testing functions example” on page 70](#)

tolower

Character conversion macro. For example usage see [“Example of tolower\(\), toupper\(\) usage.” on page 79](#).

```
#include <ctype.h>
int tolower(int c);
```

c	int	character being evaluated
---	-----	---------------------------

`tolower()` returns the lowercase equivalent of uppercase letters and returns all other characters unchanged.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isalpha” on page 71](#)
[“toupper” on page 79.](#)

Listing 8.3 Example of tolower(), toupper() usage.

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

int main(void)
{
    static char s[] =
        " ** DELICIOUS! lovely? delightful **";
    int i;

    for (i = 0; s[i]; i++)
        putchar(tolower(s[i]));
    putchar('\n');

    for (i = 0; s[i]; i++)
        putchar(toupper(s[i]));
    putchar('\n');

    return 0;
}
```

Output:
** delicious! lovely? delightful **
** DELICIOUS! LOVELY? DELIGHTFUL **

toupper

Character conversion macro.

```
#include <ctype.h>
int toupper(int c);
```

c	int	character being evaluated
---	-----	---------------------------

Remarks

The function `toupper()` returns the uppercase equivalent of a lowercase letter and returns all other characters unchanged.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isalpha” on page 71](#)

[“tolower” on page 78](#)

For example usage see [“Example of tolower\(\), toupper\(\) usage.” on page 79](#)

direct.h

The `direct.h` header file consists of various runtime declarations that pertain to the Win32 x86 targets for reading and manipulation of directories/folders.

Overview of direct.h

The `direct.h` header file consists of

- [“`getdcwd`” on page 81](#) gets the current working directory
- [“`getdiskfree`” on page 82](#) gets the amount of free disk space
- [“`getdrives`” on page 82](#) gets drives available

`getdcwd`

Determines the current working directory information.

```
#include <direct.h>
char * _getdcwd(int drive, char *path, int len);
```

drive	int	The current drive as a number
path	char *	The current working directory path
len	int	The buffer size

Remarks

If the full path exceeds the buffers length unexpected results may occur.

The current working directory is returned if successful or NULL if failure occurs.

This function is Windows only when declared from this header.

See Also

[“ chdrive” on page 101](#)

_getdiskfree

Determines the free disk space.

```
#include <direct.h>
unsigned _getdiskfree(unsigned drive, struct _diskfree_t *
    dfree);
```

drive	unsigned int	The current drive as a number
dfree	_diskfree_t *	A structure that holds the disk information

Remarks

The structure _diskfree_t holds the disk attributes.

Zero is returned on success, true value is returned on failure.

This function is Windows only when declared from this header.

See Also

[“ getdrive” on page 105](#)

_getdrives

Determines the logical drive information.

```
#include <direct.h>
unsigned long _getdrives(void);
```

Remarks

There is no parameter for this function.

Returns the logical drives as a long integer value. Bit 0 is drive A, bit 1 is drive B, bit 2 is drive C and so forth.

This function is Windows only when declared from this header.

See Also

[“_getdrive” on page 105](#)

direct.h

Overview of direct.h

dirent.h

The header `dirent.h` defines several file directory functions for reading directories.

Overview of dirent.h

The `dirent.h` header file consists of

- [“opendir” on page 85](#) opens a directory stream
- [“readdir” on page 86](#) reads a directory stream
- [“readdir_r” on page 87](#) reentrant read a directory stream
- [“rewinddir” on page 88](#) rewinds a directory stream
- [“closedir” on page 88](#) closes a directory stream

NOTE

If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

opendir

This function opens the directory named as an argument and returns a stream pointer of type `DIR`.

```
#include <dirent.h>
DIR * opendir(const char *path);
```

path	const char *	The path of the directory to be opened
------	--------------	--

Remarks

This function returns NULL if the directory can not be opened. If successful a directory stream pointer of type DIR * is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“closedir” on page 88](#)

readdir

This function is used read the next directory entry.

```
#include <dirent.h>
struct dirent * readdir(DIR *dp);
```

dp	DIR *	The stream being read
----	-------	-----------------------

Remarks

The data pointed to by readdir() may be overwritten by another call to readdir().

The function readdir returns the next directory entry from the stream dp as a pointer of struct dirent.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“rewinddir” on page 88](#)

readdir_r

This function is the reentrant version to read the next directory entry.

```
#include <dirent.h>

int readdir_r(DIR *ref, struct dirent *entry,
              struct dirent ** result);
```

dp	DIR *	The stream being read
entry	struct dirent *	
result	struct dirent **	

Remarks

The `readdir_r()` function provides the same service as [“readdir” on page 86](#). The difference is that `readdir()` would return a pointer to the next directory entry, and that pointer was internal to the library implementation. For `readdir_r()`, the caller provides the storage for the `dirent` struct.

On a successful call to `readdir_r()`, the function result is zero, the storage for entry is filled with the next directory entry, and the result pointer contains a pointer to entry. If the end of the directory is reached, the function result is zero and the result pointer is `NULL`. If any error occurs, the function result is an error code.

This function may require extra library support.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“rewinddir” on page 88](#)

[“readdir” on page 86](#)

rewinddir

This function `rewinddir` resets the directory stream to the original position.

```
#include <dirent.h>
void rewinddir(DIR * dp);
```

dp	DIR *	The stream being rewound
----	-------	--------------------------

Remarks

There is no return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“readdir” on page 86](#)

closedir

The function `closedir()` ends the directory reading process. It deallocates the directory stream pointer and frees it for future use.

```
#include <dirent.h>
int closedir(DIR * dp);
```

dp	DIR *	The directory stream pointer to be closed
----	-------	---

Remarks

The function `closedir()` returns zero on success and the value of -1 on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“opendir” on page 85](#)

dirent.h

Overview of dirent.h

div_t.h

The div.h.h header defines two structures used for math computations.

Overview of div_t.h

The div_t.h header file consists of three structures.

- [“div_t” on page 91](#), stores remainder and quotient variables
- [“ldiv_t” on page 92](#), stores remainder and quotient variables
- [“lldiv_t” on page 92](#) stores remainder and quotient variables

div_t

Stores the remainder and quotient from the div function.

```
#include <div_t.h>

typedef struct {

    int quot;
    int rem;
} div_t;
```

Remarks

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“div” on page 476](#)

div_t.h

Overview of div_t.h

ldiv_t

Stores the remainder and quotient from the `ldiv` function.

```
#include <div_t.h>

typedef struct {

    int    quot;
    int    rem;
} ldiv_t;
```

Remarks

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ldiv” on page 482](#)

lldiv_t

Stores the remainder and quotient from the `lldiv` function.

```
#include <div_t.h>

typedef struct {

    long long    quot;
    long long    rem;
} lldiv_t;
```

Remarks

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ldiv” on page 482](#)

div_t.h

Overview of div_t.h

errno.h

The `errno.h` header file provides the global error code variable `errno`.

Overview of `errno.h`

There is one global declared in `errno.h`

- [“errno” on page 95](#)

errno

The `errno.h` header file provides the global error code variable `errno`.

```
#include <errno.h>  
extern int errno;
```

NOTE

The math library used for Mac OS and Windows (when optimized) is not fully compliant with the 1990 ANSI C standard in that none of the math functions set `errno`. The MSL math libraries provide better means of error detection. Using `fpclassify` (which is C99 portable) provides a better error reporting mechanism. The setting of `errno` is considered an obsolete mechanism because it is inefficient as well as uninformative.

Most functions in the standard library return a special value when an error occurs. Often the programmer needs to know about the nature of the error. Some functions provide detailed error information by assigning a value to the global variable `errno`. The `errno` variable is declared in the `errno.h` header file. See [“Error Number Definitions” on page 96](#).

The `errno` variable is not cleared when a function call is successful; its value is changed only when a function that uses `errno` returns its own error value. It is the programmer's responsibility to assign 0 to `errno` before calling a function that uses it. For example usage see [Listing 12.2](#).

The table “[Error Number Definitions](#)” lists the error number macros defined in MSL. Not all these values are used in MSL but are defined in POSIX and other systems and are therefore defined in MSL to facilitate the compilation of codes being ported from other platforms.

This macro may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 12.1 Error Number Definitions

errno value	Description
EACCES	Permission denied
EFPOS	File Position Error
EILSEQ	Wide character encoding error
ENAMETOOLONG	File name too long
ENOENT	No such file or directory
ENOSYS	Function not implemented
ERANGE	Range error. The function cannot return a value
ESIGPARM	Signal error

Platform Assigned

EBADF	Win32 assigned only, Bad file descriptor
EINVAL	Win32 assigned only, Invalid argument
ENOERR	Win32 assigned only, Bad file number
ENOMEM	Win32 assigned only, No error detected
EMACOSERR	Mac OS assigned only, the value is assigned to the global variable __MacOSErrNo
Unassigned errno	reserved for future use
E2BIG	Argument list too long
EAGAIN	Resource temporarily unavailable
EBUSY	Device busy

Listing 12.1 Error Number Definitions

ECHILD	No child processes
EDEADLK	Resource deadlock avoided
EDOM	Numerical argument out of domain
EEXIST	File already exists
EFAULT	Bad address
EFBIG	File too large
EINTR	Interrupted system call
EIO	Input/output error
EISDIR	Is a directory
EMFILE	Too many open files
EMLINK	Too many links
ENFILE	Too many open files in system
ENODEV	Operation not supported by device
ENOEXEC	Exec format error
ENOLCK	No locks available
ENOSPC	No space left on device
ENOTDIR	Not a directory
ENOTEMPTY	Directory not empty
ENOTTY	Inappropriate ioctl for device
ENXIO	Device not configured
EPERM	Operation not permitted
EPIPE	Broken pipe
EROFS	Read-only file system
ESPIPE	Illegal seek
ESRCH	No such process
EUNKNOWN	Unknown error
EXDEV	Cross-device link

Listing 12.2 errno example

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

int main(void)
{
    char *num = "5000000000";
    long result;
    result = strtol( num, 0, 10);

    if (errno == ERANGE)
        printf("Range error!\n");
    else
        printf("The string as a long is %ld", result);

    return 0;
}
```

Output:

Range error!

extras.h

The header extras.h defines several CodeWarrior provided non standard console functions.

Overview of extras.h

The extras.h header file consists of several functions which may be indirectly included by other headers or included for use directly through extras.h.

- [“`chdrive`” on page 101](#) changes the working drive
- [“`chsize`” on page 102](#) changes a file size
- [“`filelength`” on page 102](#) gets the file length
- [“`fileno`” on page 103](#) gets the file number
- [“`fullpath`” on page 104](#) gets the full pathname
- [“`gcvt`” on page 105](#) converts a floating point value to a string
- [“`getdrive`” on page 105](#) gets the drive as a number
- [“`GetHandle`” on page 106](#) (Windows only) get console handle
- [“`get_osfhandle`” on page 106](#) get an operating system handle
- [“`heapmin`” on page 107](#) releases unused heap to the system
- [“`itoa`” on page 107](#) int to string
- [“`itow`” on page 108](#) int to wide character string
- [“`ltoa`” on page 109](#) long to string
- [“`ltow`” on page 109](#) long to wide character string
- [“`makepath`” on page 110](#) creates a path
- [“`open_osfhandle`” on page 111](#) open an operating system handle
- [“`putenv`” on page 111](#) put an environment variable
- [“`searchenv`” on page 112](#) search the environment variable
- [“`splitpath`” on page 113](#) splits a path into components

-
- “[strcasecmp](#)” on page 114 string ignore case compare
 - “[strcmpi](#)” on page 114 case insensitive string compare
 - “[strdate](#)” on page 115 stores a date in a string
 - “[strdup](#)” on page 116 duplicates a string
 - “[strcmp](#)” on page 116 case insensitive string compare
 - “[stricoll](#)” on page 117 locale collating string comparison
 - “[strlwr](#)” on page 118 string to lower case
 - “[strncasecmp](#)” on page 118 string case compare with length specified
 - “[strncmpi](#)” on page 119 case insensitive string compare
 - “[strnicmp](#)” on page 120 case insensitive string compare with length specified.
 - “[strncoll](#)” on page 120 length limited comparison of a string collated by locale setting
 - “[strnicoll](#)” on page 121 case insensitive string comparison by locale.
 - “[strnset](#)” on page 122 sets a number of characters in a string
 - “[strrev](#)” on page 123 reverses characters in a string
 - “[strset](#)” on page 123 sets characters in a set
 - “[strspnp](#)” on page 124 finds a string in another string
 - “[strupr](#)” on page 125 string to upper case
 - “[tell](#)” on page 125 gets the file indicator position
 - “[ultoa](#)” on page 127 unsigned long to string
 - “[_ultow](#)” on page 127 unsigned long to wide character string
 - “[wcsdup](#)” on page 128 wide character string duplicate
 - “[wcsicoll](#)” on page 129 case insensitive string comparison collated by locale setting
 - “[wcsicmp](#)” on page 129 wide character case insensitive string compare
 - “[wcslwr](#)” on page 130 wide character string to lower case
 - “[wcsncoll](#)” on page 131 wide character string comparison collated by locale settings
 - “[wcsnicmp](#)” on page 132 wide character case insensitive string compare for a length limit
 - “[wcsnicoll](#)” on page 131 wide character string comparison case insensitive collated by locale
 - “[wcsnset](#)” on page 133 sets a number of characters in a wide character string

- [“wcsrev” on page 134](#) reverses a wide character string
- [“wceset” on page 134](#) sets characters in a wide character string
- [“wcspnnp” on page 135](#) finds a wide character string in another wide character string
- [“wstrrev” on page 136](#) reverses wide character string
- [“wcsupr” on page 135](#) wide string to upper case
- [“wtoi” on page 136](#) wide string to integer

NOTE If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

_chdrive

Changes the current drive by number.

```
#include <extras.h>
_chdrive(int drive);
```

drive	int	The drive as a number
-------	-----	-----------------------

Remarks

The drive is listed as a number, 1 for A, 2 for B, 3 for C, and so forth.

Zero is returned on success and negative one on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“_getdrive” on page 105](#)

chsize

This function is used to change a file's size.

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

handle	int	The handle of the file being changed
size	long	The size to change

Remarks

If a file is truncated all data beyond the new end of file is lost.

This function returns zero on success and a negative one if a failure occurs.

See Also

[“GetHandle” on page 106](#)

filelength

Retrieves the file length based on a file handle.

```
#include <extras.h>
int filelength(int fileno);
int _filelength(int fileno);
```

fileno	int	The file as a handle
--------	-----	----------------------

The filelength as an int value is returned on success. A negative one is returned on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“GetHandle” on page 106](#)

fileno

Obtains the file descriptor associated with a stream.

```
#include <extras.h>
int fileno(FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

This function obtains the file descriptor for the stream. You can use the file descriptor with other functions in `unix.h`, such as `read()` and `write()`.

For the standard I/O streams `stdin`, `stdout`, and `stderr`, `fileno()` returns the following values:

Listing 13.1 Fileno return value for standard streams

This function call...	Returns this file descriptor...
<code>fileno(stdin)</code>	0
<code>fileno(stdout)</code>	1
<code>fileno(stderr)</code>	2

If it is successful, `fileno()` returns a file descriptor. If it encounters an error, it returns `-1` and sets `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fdopen” on page 348](#)

[“open, wopen” on page 143](#)

extras.h

Overview of extras.h

Figure 13.1 Example of fileno() usage.

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

int main(void)
{
    printf("The handle for the standard input device is: %d",
           fileno(stdin) );

    return 0;
}
```

Reult

The handle for the standard input device is: 0

_fullpath

Converts a relative path name to a full path name.

```
#include <extras.h>

char *_fullpath(char * absPath,
                 const char * relPath, size_t maxLength);
```

absPath	char *	The full absolute path
relPath	const char *	The relative path
maxLength	size_t	The maximum path length

Remarks

If the maxLength is `NULL` a path of up to `MAX_PATH` is used.

A pointer to the `absPath` is returned on success. On failure `NULL` is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

gcvt

This function converts a floating point value to a null terminated char string.

```
#include <extras.h>
char *gcvt(double value, int digits, char *buffer);
char *_gcvt(double value, int digits, char *buffer);
```

value	double	The floating point value to be converted into a string
digits	int	The number of significant digits to converted
buffer	char *	The string to hold the converted floating point value

Remarks

The resultant string includes the decimal point and sign of the original value.

This function returns a pointer to the `buffer` argument.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“atof” on page 465](#)

_getdrive

Finds the current drive as a number.

```
#include <extras.h>
int _getdrive();
```

This facility has no parameters.

The current drive number is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

- [“_chdrive” on page 101](#)
- [“GetHandle” on page 106](#)

GetHandle

GetHandle retrieves the current objects handle.

```
#include <extras.h>
int GetHandle();
```

This facility has no parameters.

The device handle.is returned on success. A negative one is returned on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

- [“fileno” on page 103](#)

_get_osfhandle

Retrieve an operating system file handle.

```
#include <extras.h>
long _get_osfhandle(int filehandle);
```

filehandle	int	An operating system file handle
------------	-----	---------------------------------

An operating system file handle as opposed to C file handle is returned if successful otherwise sets `errno` and returns NULL.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“GetHandle” on page 106](#)
- [“ open_osfhandle” on page 111,](#)

heapmin

This function releases the heap memory back to the system.

```
#include <extras.h>
int heapmin(void);
int _heapmin(void);
```

This facility has no parameters.

Heapmin returns zero if successful otherwise sets errno to ENOSYS and returns -1;

itoa

This function converts an int value to a null terminated char array.

```
#include <extras.h>
char * itoa(int val, char *str, int radix);
char * _itoa(int val, char *str, int radix);
```

val	int	The integer value to convert
str	char *	The string to store the converted value
radix	int	The numeric base of the number to be converted

Remarks

The radix is the base of the number in a range of 2 to 36.

A pointer to the converted string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“atoi” on page 466](#)

[“itoa” on page 109](#)

itow

This function converts an int value to a null terminated wide char array.

```
#include <extras.h>
wchar_t* itow(int val, wchar_t *str, int radix);
wchar_t* _itow(int val, wchar_t *str, int radix);
```

val	int	The integer value to convert
str	wchar_t *	The string to store the converted value
radix	int	The numeric base of the number to be converted

Remarks

The radix is the base of the number in a range of 2 to 36.

A pointer to the converted wide character string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“itoa” on page 107](#)

Itoa

This function converts a long int value to a null terminated char array.

```
#include <extras.h>

#define ltoa(x, y, z) _itoa(x, y, z);
#define _ltoa(x, y, z) _itoa(x, y, z);
```

Remarks

This function simply redefines _itoa for 32 bit systems.

A pointer to the converted wide character string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“itoa” on page 107](#)

[“atol” on page 467](#)

_ltoiw

This function converts an long value to a null terminated wide char array.

```
#include <extras.h>

wchar_t *_ltoiw(unsigned long val, wchar_t *str, int radix);
```

val	unsigned long	The long value to convert
val	wchar_t *	The wide character string to store the converted value
radix	int	The numeric base of the number to be converted

Remarks

The radix is the base of the number in a range of 2 to 36.

A pointer to the converted wide character string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“itow” on page 108,](#)

[“ltoa” on page 109,](#)

makepath

Makepath is used to create a path.

```
#include <extras.h>
void makepath(char *path, const char *drive,
              const char *dir, const char *fname, const char *ext);
void _makepath(char *path, const char *drive,
               const char *dir, const char *fname, const char *ext);
```

path	char *	String to receive the created path
drive	const char *	String containing the drive component
dir	const char *	String containing the directory component
fname	const char *	String containing the file name component
ext	const char *	String containing the file extension component

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“chdrive” on page 101](#)

_open_osfhandle

Opens an operating system handle.

```
#include <extras.h>
int _open_osfhandle(long ofshandle, int flags);
```

ofshandle	long	The file as an operating system handle
flags	int	file opening flags

Remarks

This function opens an operating system handle as a C file handle.

The file handle value is returned on success. A negative one is returned on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“_get_osfhandle” on page 106.](#)

putenv

Adds a string to the environmental variable.

```
#include <extras.h>
int putenv(const char * inVarName)
int _putenv(const char * inVarName)
```

inVarName	char *	The string to add to the environmental variable
-----------	--------	---

Remarks

May also be used to modify or delete an existing string. The string must be a global value.

The environment is restored at the program termination.

Zero is returned on success or negative one on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

_searchenv

Searches the environmental path for a file.

```
#include <extras.h>
void _searchenv(const char *filename,
               const char *varname, char *pathname);
```

filename	const char *	The file to search for
varname	const char *	The environmental variable name
pathname	char *	The path name of the file

Remarks

The current drive is searched first then a specified environmental variable path is searched.

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

splitpath

This function takes a path and returns pointers to each of the components of the path.

```
#include <extras.h>

void splitpath (const char *path, char *drive,
                char *dir, char *fname, char *ext)
void _splitpath (const char *path, char *drive,
                 char *dir, char *fname, char *ext)
```

path	const char *	The file path to be split
drive	char *	The string to receive the drive component
dir	char *	The string to receive the directory component
fname	char *	The string to receive the filename component
ext	char *	The string to receive the file extension component

Remarks

The programmer must provide arrays large enough to hold the various components

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“chdir” on page 577](#)

strcasecmp

Ignore case string comparison function

```
#include <extras.h>
int strcasecmp (const char *str1, const char *str2);
```

str1	const char *	String being compared
str2	const char *	Comparison string

Remarks

The function converts both strings to lower case before comparing them.

Strcasecmp returns greater than zero if str1 is larger than str2 and less than zero if str2 is larger than str1. If they are equal returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strncasecmp” on page 118](#)

[“strcmp” on page 116.](#)

strcmpi

A case insensitive string compare.

```
#include <extras.h>
int strcmpi(const char *s1, const char *s2);
int _strcmpi(const char *s1, const char *s2);
```

s1	const char *	The first string to compare
s2	const char *	The comparison string

An integer value of less than zero if the first argument is less than the second in a case insensitive string comparison. A positive value if the first argument is greater than the second argument in a case insensitive string comparison. Zero is returned if both case insensitive strings are the same.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strcasecmp” on page 114](#)
- [“strnicmp” on page 120](#)

strdate

The strdate function stores a date in a buffer provided.

```
#include <extras.h>
char * strdate(char *str);
char * _strdate(char *str);
```

str	char *	A char string to store the date
-----	--------	---------------------------------

The function returns a pointer to the `str` argument

Remarks

This function stores a date in the buffer in the string format of mm/dd/yy where the buffer must be at least 9 characters.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strftime” on page 562](#)

strup

Creates a duplicate string in memory.

```
#include <extras.h>
char * strdup(const char *str);
char * _strdup(const char *str);
```

str	const char *	The string to be copied
-----	--------------	-------------------------

A pointer to the storage location or NULL if unsuccessful.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memcpy” on page 516](#)

strcmp

A function for string comparison ignoring case.

```
#include <extras.h>
int strcmp(const char *s1,const char *s2);
int _strcmp(const char *s1,const char *s2);
```

s1	const char *	The string being compared
s2	const char *	The comparison string

Stricmp returns greater than zero if str1 is larger than str2 and less than zero if str2 is larger than str 1. If they are equal returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcmp” on page 520](#)
[“strncmp” on page 531](#)

stricoll

A case insensitive locale collating string comparison.

```
#include <extras.h>

int stricoll(const char *s1, const char *s2);
int _stricoll(const char *s1, const char *s2);
```

s1	const char *	The string to compare
s2	const char *	A comparison string

Remarks

The comparison is done according to a collating sequence specified by the `LC_COLLATE` component of the current locale.

If the first string is less than the second a negative number is returned. If the first string is greater than the second then a positive number is returned. If both strings are equal then zero is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strncoll” on page 120](#)
[“wcsicoll” on page 129](#)

strlwr

This function converts a string to lowercase.

```
#include <extras.h>
char * strlwr(char *str);
char * _strlwr(char *str);
```

str	char	The string being converted
-----	------	----------------------------

A pointer to the converted string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strupr” on page 125](#)

[“tolower” on page 78](#)

strncasecmp

Ignore case string comparison function with length specified.

```
#include <string.h>
int strncasecmp(const char *s1, const char *s2, unsigned n);
```

str1	const char *	String being compared
str2	const char *	Comparison string
n	unsigned int	Length of comparison

Remarks

The function converts both strings to lower case before comparing them.

Strncasecmp returns greater than zero if str1 is larger than str2 and less than zero if str2 is larger than str 1. If they are equal returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strcasecmp” on page 114](#)
- [“strcmp” on page 120,](#)

strncMPI

A length limited case insensitive comparison of one string to another string.

```
#include <extras.h>

int strncMPI(const char *s1, const char *s2, size_t n);
int _strncMPI(const char *s1, const char *s2, size_t n);
```

s1	const char *	A string to compare
s2	const char *	A comparison string
n	size_t	number of characters to compare

Remarks

Starting at the beginning of the string characters of the strings are compared until a difference is found or until `n` characters have been compared.

If the first argument is less than the second argument in a case insensitive comparison a negative integer is returned. If the first argument is greater than the second argument in a case insensitive comparison then a positive integer is returned. If they are equal then zero is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strcmp” on page 116](#)
- [“strcasecmp” on page 118](#)
- [“wcsicmp” on page 129](#)

strncoll

A length limited locale collating string comparison.

```
#include <extras.h>

int strncoll(const char *s1, const char *s2, size_t n);
int _strncoll(const char *s1, const char *s2, size_t sz);
```

s1	const char *	The string to compare
s2	const char *	A comparison string
sz	size_t	Number of characters to collate

Remarks

The comparison is done according to a collating sequence specified by the `LC_COLLATE` component of the current locale.

If the first string is less than the second a negative number is returned. If the first string is greater than the second then a positive number is returned. If both strings are equal then zero is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“strnicoll” on page 121](#)

[“wcsncoll” on page 131](#)

strnicmp

A function for string comparison ignoring case but specifying the comparison length.

```
#include <extras.h>

int strnicmp(const char *s1, const char *s2, int n);
int _strnicmp(const char *s1, const char *s2, int n);
```

s1	const char *	The string being compared
s2	const char *	The comparison string
n	int	Maximum comparison length

The function strnicmp returns greater than zero if s1 is larger than s2 and less than zero if s2 is larger than s1. If they are equal returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcmp” on page 520](#)

[“strncmp” on page 531](#)

strnicoll

A case insensitive locale collating string comparison with a length limitation.

```
#include <extras.h>

int strnicoll(const char *s1, const char *s2, size_t sz);
int _strnicoll(const char *s1, const char *s2, size_t sz);
```

s1	const char *	The string to compare
s1	const char *	A comparison string
sz	size_t	

Remarks

The comparison is done according to a collating sequence specified by the LC_COLLATE component of the current locale.

If the first string is less than the second a negative number is returned. If the first string is greater than the second then a positive number is returned. If both strings are equal then zero is returned.

See Also

[“stricoll” on page 117](#)

[“wcsicoll” on page 129](#)

strnset

This function sets the first n characters of string to a character.

```
#include <extras.h>
char * strnset(char *str, int c, size_t n)
char * _strnset(char *str, int c, size_t n)
```

str	char *	The string to be modified
c	int	The char to be set
n	size_t	The number of characters of str to be set to the value of c

Remarks

If the number of characters exceeds the length of the string all characters are set except for the terminating character.

A pointer to the altered string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“strset” on page 123](#)

strrev

This function reverses a string.

```
#include <extras.h>
char * _strrev(char *str);
char * _strrev(char *str);
```

str	char *	The string to be reversed
-----	--------	---------------------------

A pointer to the reversed string.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcpy” on page 524](#)

strset

This function sets characters of string to a character

```
#include <extras.h>
char * strset(char *str, int c)
char * _strset(char *str, int c)
```

str	char *	The string to be modified
c	int	The char to be set

A pointer to the altered string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strnset” on page 122](#)

strspnp

This function returns pointer to first character in s1 that isn't in s2

```
#include <extras.h>
char * strspnp(char *s1, const char *s2)
char * _strspnp(char *s1, const char *s2)
```

s1	char *	The string being checked
s2	const char *	The search string as a char set.

Remarks

This function determines the position in the string being searched is not one of the chars in the string set.

A pointer to the first character in s1 that is not in s2 or NULL is returned if all the characters of s1 are in s2.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcspn” on page 525](#)

[“strspn” on page 536](#)

strupr

The function `strupr` converts a string to uppercase.

```
#include <extras.h>
char * strupr(char *str);
char * _strupr(char *str);
```

str	char	The string being converted
-----	------	----------------------------

A pointer to the converted string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“toupper” on page 79](#)

[“strlwr” on page 118](#)

tell

Returns the current offset for a file.

```
#include <extras.h>
long tell(int fildes);
```

fildes	int	The file descriptor
--------	-----	---------------------

Remarks

This function returns the current offset for the file associated with the file descriptor `fildes`. The value is the number of bytes from the file's beginning.

If it is successful, `tell()` returns the offset. If it encounters an error, `tell()` returns `-1L`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“ftell” on page 391](#)

[“lseek” on page 594](#)

Listing 13.2 Example of tell() usage.

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

int main(void)
{
    int fd;
    long int pos;

    fd = open("mytest", O_RDWR | O_CREAT | O_TRUNC);
    write(fd, "Hello world!\n", 13);
    write(fd, "How are you doing?\n", 19);

    pos = tell(fd);

    printf("You're at position %ld.", pos);

    close(fd);

    return 0;
}
```

Result

This program prints the following to standard output:
You're at position 32.

ultoa

This function converts an unsigned long value to a null terminated char array.

```
#include <extras.h>

char * ultoa(unsigned long val, char *str, int radix);
char * _ultoa(unsigned long val, char *str, int radix);
```

val	unsigned long	The integer value to convert
str	char *	The string to store the converted value
radix	int	The numeric base of the number to be converted

Remarks

The radix is the base of the number in a range of 2 to 36. This function is the converse of `strtoul()` and uses the number representation described there.

A pointer to the converted string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ltoa” on page 109](#)

[“itoa” on page 107](#)

_ultow

This function converts an unsigned long value to a null terminated wide character array.

```
#include <extras.h>

wchar_t *_ultow(unsigned long val, wchar_t *str, int radix);
```

val	unsigned long	The value to be converted
str	wchar_t *	A buffer large enough to hold the converted value
radix	int	The base of the number being converted

Remarks

The radix is the base of the number in a range of 2 to 36. This function is the wide character equivalent of `strtoul()` and uses the number representation described there.

A pointer to the converted wide character string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strtoul” on page 502](#)

[“itow” on page 108](#)

[“_ltow” on page 109](#)

wcsdup

Creates a duplicate wide character string in memory.

```
#include <extras.h>
wchar_t * wcsdup (const wchar_t *str)
wchar_t * _wcsdup (const wchar_t *str)
```

str	const wchar_t *	The string to be copied
-----	-----------------	-------------------------

A pointer to the storage location is returned upon success or NULL is returned if unsuccessful.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strdup” on page 116](#)

wcsicmp

A function for wide character string comparison ignoring case.

```
#include <extras.h>
int wcsicmp(const wchar_t *s1, const wchar_t *s2)
int _wcsicmp(const wchar_t *s1, const wchar_t *s2)
```

s1	const wchar_t *	The string being compared
s2	const wchar_t *	The comparison string

The function _wcsicmp returns greater than zero if str1 is larger than str2 and less than zero if str2 is larger than str1. If they are equal returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcmp” on page 520](#)

[“strncmp” on page 531](#)

wcsicoll

A case insensitive locale collating wide character string comparison.

```
#include <extras.h>
int wcsicoll(const wchar_t *s1, const wchar_t *s2);
int _wcsicoll(const wchar_t *s1, const wchar_t *s2);
```

s1	const wchar_t *	The wide string to compare
s2	const wchar_t *	A comparison wide character string

Remarks

The comparison is done according to a collating sequence specified by the `LC_COLLATE` component of the current locale.

If the first string is less than the second a negative number is returned. If the first string is greater than the second then a positive number is returned. If both strings are equal then zero is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“wcsncoll” on page 131](#)

[“stricoll” on page 117](#)

wcslwr

This function converts a string to lowercase.

```
#include <extras.h>
wchar_t *wcslwr (wchar_t *str);
wchar_t *_wcslwr (wchar_t *str);
```

str	wchar_t	The string being converted
-----	---------	----------------------------

A pointer to the converted string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“strupr” on page 125](#)

[“strlwr” on page 118](#)

wcsncoll

A length limited locale collating wide string comparison.

```
#include <extras.h>

int wcsncoll(const wchar_t *s1, const wchar_t *s2, size_t sz);
int _wcsncoll(const wchar_t *s1, const wchar_t *s2, size_t sz);
```

s1	const wchar_t *	The wide string to compare
s2	const wchar_t *	A comparison wide character string
sz	size_t	

Remarks

The comparison is done according to a collating sequence specified by the `LC_COLLATE` component of the current locale.

If the first string is less than the second a negative number is returned. If the first string is greater than the second then a positive number is returned. If both strings are equal then zero is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“stricoll” on page 117](#)

[“wcsnicoll” on page 131](#)

wcsnicoll

A length limited locale collating wide string case insensitive comparison.

```
#include <extras.h>

int wcsnicoll(const wchar_t *s1, const wchar_t *s2, size_t sz);
int _wcsnicoll(const wchar_t *s1, const wchar_t *s2, size_t sz);
```

s1	const wchar_t *	The wide string to compare
s2	const wchar_t *	A comparison wide character string
sz		

Remarks

The comparison is done according to a collating sequence specified by the `LC_COLLATE` component of the current locale.

If the first string is less than the second a negative number is returned. If the first string is greater than the second then a positive number is returned. If both strings are equal then zero is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wcsncoll” on page 131](#)

[“strnicoll” on page 121](#)

wcsnicmp

A function for a wide character string comparison ignoring case but specifying the comparison length.

```
#include <extras.h>
iint wcsnicmp(const wchar_t *s1, const wchar_t *s2, size_t n);
iint _wcsnicmp(const wchar_t *s1, const wchar_t *s2, size_t n);
```

s1	const wchar_t *	The string being compared
s2	const wchar_t *	The comparison string
n	int	Maximum comparison length

The function `_wcsnicmp` returns greater than zero if str1 is larger than str2 and less than zero if str2 is larger than str 1. If they are equal returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“stricmp” on page 116](#)
- [“strncmp” on page 531](#)

wcsnset

This function sets the first n characters of wide character string to a character.

```
#include <extras.h>
wchar_t *wcsnset(wchar_t *str, wchar_t wc, size_t n);
wchar_t *_wcsnset(wchar_t *str, wchar_t wc, size_t n);
```

str	wchar_t *	The string to be modified
wc	wchar_t	The char to be set
n	size_t	The number of characters in the string to set.

Remarks

If the number of characters exceeds the length of the string all characters are set except for the terminating character.

A pointer to the altered string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strset” on page 123](#)

extras.h*Overview of extras.h***wcsrev**

This function reverses a wide character string.

```
#include <extras.h>
wchar_t * wcsrev(wchar_t *str);
wchar_t * _wcsrev(wchar_t *str);
```

str	wchar_t *	The string to be reversed
-----	-----------	---------------------------

A pointer to the reversed string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wstrrev” on page 136](#)

wcsset

This function sets characters of a wide character string to a wide character.

```
#include <extras.h>
wchar_t * wcsset(wchar_t *str, wchar_t wc);
wchar_t *_wcsset(wchar_t *str, wchar_t wc);
```

str	wchar_t *	The string to be modified
c	wchar_t	The char to be set

A pointer to the altered string is returned.

See Also

[“strnset” on page 122](#)

wcsspnp

This function returns pointer to first character in s1 that isn't in s2

```
#include <extras.h>
wchar_t *wcsspnp(const wchar_t *s1, const wchar_t *s2);
wchar_t *_wcsspnp(const wchar_t *s1, const wchar_t *s2);
```

s1	wchar_t *	The string being checked
s2	const wchar_t *	The search string as a char set.

Remarks

This function determines the position in the string being searched is not one of the chars in the string set.

A pointer to the first character in s1 that is not in s2 or NULL if all the characters of s1 are in s2.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strspnp” on page 124](#)

[“strspn” on page 536](#)

wcsupr

The function `_wcsupr` converts a wide character string to uppercase.

```
#include <extras.h>
wchar_t * wcsupr (wchar_t *str);
wchar_t *_wcsupr (wchar_t *str);
```

str	wchar_t *	The string being converted
-----	-----------	----------------------------

A pointer to the converted string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strupr” on page 125](#)

[“strlwr” on page 118](#)

wstrrev

This function reverses a wide character string.

```
#include <string.h>
wchar_t * _wstrrev(wchar_t * str);
wchar_t * _wstrrev(wchar_t * str);
```

str	wchar_t *	The string to be reversed
-----	-----------	---------------------------

A pointer to the reversed string is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strrev” on page 123](#)

wtoi

This function converts a null terminated wide char array to an int value.

```
#include <extras.h>
int wtoi (const wchar_t *str);
int _wtoi (const wchar_t *str);
```

str	const wchar_t *	The string to be converted to an int.
-----	-----------------	---------------------------------------

Remarks

The `_wtoi()` function converts the character array pointed to by `nptr` to an integer value.

This function skips leading white space characters.

This function sets the global variable `errno` to `ERANGE` if the converted value cannot be expressed as a value of type `int`.

The function `_wtoi()` returns an the converted integer value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“atoi” on page 466](#)

extras.h

Overview of extras.h

fcntl.h

The header file `fcntl.h` contains several file control functions that are useful for porting a program from UNIX.

Overview of `fcntl.h`

The header `fcntl.h` includes the following functions:

- [“creat, wcreate” on page 140](#) for creating a file
- [“fcntl” on page 141](#) for file control descriptor
- [“open, wopen” on page 143](#) for opening a file

fcntl.h and UNIX Compatibility

The header file `fcntl.h.h` contains several functions that are useful for porting a program from UNIX. These functions are similar to the functions in many UNIX libraries. However, since the UNIX and Macintosh operating systems have some fundamental differences, they cannot be identical. The descriptions of the functions tell you what the differences are.

Generally, you don’t want to use these functions in new programs. Instead, use their counterparts in the native API.

NOTE	If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.
-------------	---

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

creat, _wcreate

Create a new file or overwrite an existing file and a wide character variant.

```
#include <fcntl.h>

int creat(const char *filename, int mode);
int _creat(const char *filename, int mode);
int _wcreat(const wchar_t *filename, int mode);
```

filename	char *	The name of the file being created
wfilename	wchar_t *	The name of the file being created
mode	int	The open mode

Remarks

This function creates a file named `filename` you can write to. If the file does not exist, `creat()` creates it. If the file already exists, `creat()` overwrites it. The function ignores the argument `mode`.

This function call:

```
creat(path, mode);
```

is equivalent to this function call:

```
open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);
```

If it's successful, `creat()` returns the file description for the created file. If it encounters an error, it returns `-1`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fopen” on page 361](#)

[“fdopen” on page 348](#)

[“close” on page 579.](#)

Listing 14.1 Example of creat() usage.

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

int main(void)
{
    int fd;

    fd = creat("Jeff:Documents:mytest", 0);
    /* Creates a new file named mytest in the folder
       Documents on the volume Akbar. */

    write(fd, "Hello world!\n", 13);
    close(fd);
    return 0;
}
```

fcntl

Manipulates a file descriptor.

```
#include <fcntl.h>

int fcntl(int fildes, int cmd, ...);
int _fcntl(int fildes, int cmd, ...);
```

fildes	int	The file descriptor
cmd	int	A command to the file system
...		A variable argument list

Remarks

This function performs the command specified in `cmd` on the file descriptor `fildes`.

In the Metrowerks ANSI library, `fcntl()` can perform only one command, `F_DUPFD`. This command returns a duplicate file descriptor for the file that `fildes` refers to. You must include a third argument in the function call. The

new file descriptor is the lowest available file descriptor that is greater than or equal to the third argument.

Listing 14.2 Floating point characteristics

Mode	Description
F_DUPFD	Return a duplicate file descriptor.

If it is successful, `fcntl()` returns a file descriptor. If it encounters an error, `fcntl()` returns -1.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fileno” on page 103](#)

[“open, wopen” on page 143](#)

[“fdopen” on page 348.](#)

Listing 14.3 Example of fcntl() usage.

```
#include <unistd.h>

int main(void)
{
    int fd1, fd2;

    fd1 = open("mytest", O_WRONLY | O_CREAT);

    write(fd1, "Hello world!\n", 13);
    /* Write to the original file descriptor. */

    fd2 = fcntl(fd1, F_DUPFD, 0);
    /* Create a duplicate file descriptor. */

    write(fd2, "How are you doing?\n", 19);
    /* Write to the duplicate file descriptor. */

    close(fd2);

    return 0;
}
```

ReslutAfter you run this program,
the file mytest contains the following:
Hello world!
How are you doing?

open, _wopen

Opens a file and returns its id and a wide character variant.

```
#include <fcntl.h>

int open(const char *path, int oflag);
int _open(const char *path, int oflag);
```

path	char *	The file path as a string
oflag	int	The open mode

Remarks

The function `open()` opens a file for system level input and output. and is used with the UNIX style functions `read()` and `write()`.

Listing 14.4 Floating point characteristics

Mode	Description
O_RDWR	Open the file for both read and write
O_RDONLY	Open the file for read only
O_WRONLY	Open the file for write only
O_APPEND	Open the file at the end of file for appending
O_CREAT	Create the file if it doesn't exist
O_EXCL	Do not create the file if the file already exists.
O_TRUNC	Truncate the file after opening it.
O_NRESOLVE	Don't resolve any aliases.
O_ALIAS	Open alias file (if the file is an alias).
O_RSRC	Open the resource fork
O_BINARY	Open the file in binary mode (default is text mode).

`open()` returns the file id as an integer value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“close” on page 579](#)
- [“lseek” on page 594](#)
- [“read” on page 595](#)
- [“write” on page 604](#)

Listing 14.5 Example of open() usage:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#define SIZE FILENAME_MAX
#define MAX 1024

char fname[SIZE] = "DonQ.txt";

int main(void)
{
    int fdes;
    char temp[MAX];
    char *Don = "In a certain corner of la Mancha, the name of\n\
which I do not choose to remember,...";
    char *Quixote = "there lived\\none of those country\\
gentlemen, who adorn their\\nhalls with rusty lance\\
and worm-eaten targets.";

    /* NULL terminate temp array for printf */
    memset(temp, '\0', MAX);

    /* open a file */
    if((fdes = open(fname, O_RDWR | O_CREAT ))== -1)
    {
        perror("Error ");
        printf("Can not open %s", fname);
        exit( EXIT_FAILURE );
    }

    /* write to a file */
    if( write(fdes, Don, strlen(Don)) == -1)
    {
        printf("%s Write Error\n", fname);
        exit( EXIT_FAILURE );
    }

    /* move back to over write ... characters */
    if( lseek( fdes, -3L, SEEK_CUR ) == -1L)
    {
```

fcntl.h

Overview of fcntl.h

```
    printf("Seek Error");
    exit( EXIT_FAILURE );
}

/* write to a file */
if( write(fdes, Quixote, strlen(Quixote)) == -1)
{
    printf("Write Error");
    exit( EXIT_FAILURE );
}

/* move to beginning of file for read */
if( lseek( fdes, 0L, SEEK_SET ) == -1L)
{
    printf("Seek Error");
    exit( EXIT_FAILURE );
}

/* read the file */
if( read( fdes, temp, MAX ) == 0)
{
    printf("Read Error");
    exit( EXIT_FAILURE );
}

/* close the file */
if(close(fdes))
{
    printf("File Closing Error");
    exit( EXIT_FAILURE );
}

puts(temp);

return 0;
}
```

In a certain corner of la Mancha, the name of which I do not choose to remember, there lived one of those country gentlemen, who adorn their halls with rusty lance and worm-eaten targets.

fcntl.h

Overview of fcntl.h

fenv.h

The `<fenv.h>` header file prototypes and defines C99 data types, macros and functions that allow interaction with the floating-point environment.

Overview of fenv.h

Using the data types, macros, and functions in `fenv.h` programmers are able to test and change rounding direction as well as test, set and clear exception flags. Both the rounding direction and exception flags can be saved and restored as a single entity.

Data Types

There are two data types defined in `fenv.h`

- [“`fenv_t`” on page 149](#), the floating point environment type
- [“`fexcept_t`” on page 149](#), the floating point exception type

fenv_t

This type represents the entire floating-point environment.

fexcept_t

The type represents the floating-point exception flags collectively.

Macros

Each macro correlates to a unique bit position in the floating-point control register and a bitwise OR of any combination of macros results in distinct values. Macro values are platform dependent. There are three distinct macro types.

- [“Floating-point exceptions” on page 151,](#)
- [“Rounding Directions” on page 150,](#)
- [“Environment” on page 151,](#)

Floating-Point Exception Flags

FE_DIVBYZERO	Divide by zero
FE_INEXACT	Inexact value
FE_INVALID	Invalid value
FE_OVERFLOW	Overflow value
FE_UNDERFLOW	Underflow value
FE_ALL_EXCEPT	The result of a bitwise OR of all the floating-point exception macros

Rounding Directions

FE_DOWNWARD	Rounded downwards
FE_TONEAREST	Rounded to nearest
FE_TOWARDZERO	Rounded to zero
FE_UPWARD	Rounded upwards

Environment

FE_DFL_ENV	is a pointer to the default floating-point environment defined at the start of program execution.
------------	---

Pragmas

The header fenv.h requires one pragma “[FENV_ACC](#)” on page 151, which must be set in order for floating point flags to be tested.

FENV_ACC

FENV_ACCESS must be set to the on position in order for the floating-point flags to be tested. Whether this pragma is on or off by default is implementation dependent.

This pragma may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

```
#pragma STDC FENV_ACCESS on|off|default
```

Floating-point exceptions

The header `fenv.h` includes several floating point exception flags manipulators.

- “[feclearexcept](#)” on page 152,
- “[fegetexceptflag](#)” on page 153,
- “[feraiseexcept](#)” on page 154,
- “[fesetexceptflag](#)” on page 155,
- “[fetestexcept](#)” on page 156.

feclearexcept

The feclearexcept clears one or more floating-point exception flag indicated by its argument. The argument represents the floating-point exception flags to be cleared.

```
#include <fenv.h>
void feclearexcept(int excepts);
```

excepts	int	Determines which floating-point exception flags to clear
---------	-----	--

Remarks

The following example illustrates how programmers might want to "overlook" a floating-point exception. In this case two division operations are taking place. The first uses real numbers, the second imaginary. Suppose that even if the first operation fails we would still like to use the results from the second operation and act like no exceptions have occurred.

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 15.1 Example or feclearexcept usage

```
void ie_feclearexcept(void)
{
    float complex1[2] = {0,1};
    float complex2[2] = {0,2};
    float result[2]   = {0,0};
    feclearexcept(FE_ALL_EXCEPT);

    /* CALCULATE */
    result[0] = complex1[0] / complex2[0]; /* OOPs 0/0, sets the FE_INVALID
flag */
    result[1] = complex1[1] / complex2[1]; /* = some # */
    /* CHECK IF @ LEAST 1 OPERATION WAS SUCCESS */
    if ( (result[0] != 0) || (result[1] != 0) )
        feclearexcept(FE_INVALID);
    /* clear flag */
    else
        cout << "what ever\n";
    cout << " Rest of code ... \n";
}
```

fegetexceptflag

The fegetexceptflag function stores a representation of the states of the floating-point exception flags in the object pointed to by the argument flag. Which exception flags to save is indicated by a second argument.

```
#include <fenv.h>
void fegetexceptflag(fexcept_t *flagp, int excepts);
```

flagp	fexcept_t	Pointer to floating-point exception flags
excepts	int	Determines which floating-point exception flags to save

Remarks

The purpose of this function is to save specific floating-point exception flags to memory. In the case of the example below the saved values determine the output of the program.

Which exception flags to save is indicated by the argument excepts.

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

Listing 15.2 Example of fegetexceptflag

```
void ie_fegetexceptflag()
{
    int result = 0;
    fexcept_t flag;
    feclearexcept(FE_ALL_EXCEPT);

    /* SOME OPERATION TAKES PLACE */
    result = 1+1;
    /* NEED TO KNOW IF OPERATION WAS SUCCESSFUL */
    fegetexceptflag(&flag, FE_INVALID | FE_OVERFLOW);
    /* NOW CHECK OBJECT POINTED 2 BY flag */
    if (flag == FE_INVALID)
        cout << "The operation was invalid!\n";
    if (flag == FE_OVERFLOW)
        cout << "The operation overflowed!\n";
    if (flag == FE_INVALID | FE_OVERFLOW)
        cout << "A failure occurred\n";
    else
        cout << "success!\n";
}
```

feraiseexcept

The feraiseexcept function raises the floating-point exceptions represented by its argument.

```
#include <fenv.h>

void feraiseexcept(int excepts);
```

excepts	int	determines which floating-point exception flags to raise
---------	-----	--

Remarks

The difference between `feraiseexcept` and `fesetexceptflag` is what value the floating-point exception is raised to. `feraiseexcept` simply raises the exception, raise meaning setting to a logical one. On the other hand `fesetexceptflag` looks at the value of a stored floating-point exception flag and raises the current flag to the level of the stored value. If the stored exception flag is zero, the current exception flag is changed to a zero.

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 15.3 Example of `feraiseexcept`

```
void ie_feraiseexcept(void)
{
    feclearexcept(FE_ALL_EXCEPT); /* all exception flags = 0 */
    /* RAISE SPECIFIC FP EXCEPTION FLAGS */
    feraiseexcept(FE_INVALID | FE_OVERFLOW);
}
```

fesetexceptflag

The `fesetexceptflag` function will set the floating-point exception flags indicated by the second argument to the states stored in the object pointed to in the first argument.

```
#include <fenv.h>
void fesetexceptflag(const fexcept_t *flapp, int excepts);
```

<code>flag</code>	<code>const fexcept_t *</code>	Constant pointer to floating-point exception flags
<code>excepts</code>	<code>int</code>	Determines which floating-point exception flags to raise

Remarks

The example below illustrates how this function can be used to restore floating-point exception flags after a function call.

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 15.4 Example of fesetexceptflag

```
void ie_fesetexceptflag(void)
{
    float result = 0;
    fexcept_t flag = 0; /* fp exception flags saved here */
    feclearexcept(FE_ALL_EXCEPT);

    fegetexceptflag(&flag, FE_INVALID | FE_OVERFLOW); /* save some flags
*/
    /* SOME FUNCTION CALL */
    result = 0/0; /* OOPS caused an exception */
    for (int i = 0; i < 100; i++)
        cout << i << endl;
    /* NOW WE'RE BACK & WANT ORIGINAL VALUES OF FLAGS */
    fesetexceptflag(&flag, FE_INVALID | FE_OVERFLOW); /* restore flags
*/
}
```

fetestexcept

Use the fetestexceptflag function to find out if a floating-point exception has occurred. The argument determines which exceptions to check for.

```
#include <fenv.h>
void fetestexcept(int excepts);
```

excepts	int	Determines which floating-point exception flags to check
---------	-----	--

Remarks

The example below is similar to the fegetexceptflag example. This time though fetestexcept provides a direct interface to the exception flags so there is no need to save and then operate on the exception flag values.

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 15.5 Example of fetestexcept

```
void ie_fetestexcept(void)
{
    int result = 0;
    feclearexcept(FE_ALL_EXCEPT);

    /* SOME OPERATION TAKES PLACE */
    result = 1+1;
    /* NEED TO KNOW IF OPERATION WAS SUCCESSFUL */
    /* but instead of getting flags, check them directly */
    if (fetestexcept(FE_INVALID) == 1)
        cout << "The operation was invalid!\n";
    if (fetestexcept(FE_OVERFLOW) == 1)
        cout << "The operation overflowed!\n";
    if (fetestexcept(FE_INVALID | FE_OVERFLOW) == 1)
        cout << "A failure occurred\n";
    else
        cout << "success!\n";
}
```

Rounding

The header `fenv.h` includes two functions for determining the rounding direction.

- [“fegetround” on page 158,](#)
- [“fesetround” on page 159.](#)

fegetround

The fegetround function returns the value of the rounding direction macro representing the current rounding direction.

```
#include <fenv.h>
int fegetround(void);
```

This facility has no parameters.

Remarks

The example that follows changes the rounding direction to compute an upper bound for the expression, then restores the previous rounding direction. This example illustrates the functionality of all the rounding functions.

The function fegetround returns the current rounding direction as an integer value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 15.6 Example of fegetround

```
void ie_fegetround(void)
{
    int direction = 0;
    double x_up = 0.0, a = 5.0, b = 2.0, c = 3.0,
          d = 6.0, f = 2.5, g = 0.5, ubound = 0;
    feclearexcept(FE_ALL_EXCEPT);

    /* CALCULATE DENOMINATOR */
    fesetround(FE_DOWNWARD);
    x_up = f + g;
    /* calculate denominator */
    /* CALCULATE EXPRESSION */
    direction = fegetround();
    /* save rounding direction */
    fesetround(FE_UPWARD);
    /* change rounding direction */
    ubound = (a * b + c * d) / x_up;
    /* result should = 9.3333 */
    fesetround(direction);
    /* return original state */
    cout << " (a * b + c * d) / (f + g) = " << ubound << endl;
}
```

fesetround

The fesetround function sets the rounding direction specified by its argument, round. If the value of round does not match any of the rounding macros, the function returns 0 and the rounding direction is not changed.

```
#include <fenv.h>
int fesetround (int round);
```

round	int	Determines the direction result are rounded.
-------	-----	--

Remarks

For a function to be reentrant the function must save the rounding direction in a local variable with fegetround() and restore with fesetround() upon exit.

Zero is returned if and only if the argument is not equal.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

For an example of fesetround refer to [“Example of fegetround” on page 159](#).

Environment

The header `fenv.h` includes several functions for determining the floating-point environment information.

- [“fegetenv” on page 160](#),
- [“feholdexcept” on page 161](#),
- [“fesetenv” on page 162](#),
- [“feupdateenv” on page 163](#).

fegetenv

The fegetenv stores the current floating-point environment in the object pointed to by the argument.

```
#include <fenv.h>
void fegetenv(fenv_t *envp);
```

envp	fenv_t *	Pointer to floating-point environment
------	----------	---------------------------------------

Remarks

This function is used when a programmer wants to save the current floating-point environment, that is the state of all the floating-point exception flags and rounding direction. In the example that follows the stored environment is used to hide any floating-point exceptions raised during an interim calculation.

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 15.7 Example of fegetenv

```
long double ie_fegetenv(void)
{
    float x=0, y=0;
    fenv_t env1 =0, env2 = 0;
    feclearexcept(FE_ALL_EXCEPT);

    fesetenv(FE_DFL_ENV); /* set 2 default */
    x = x +y; /* fp op, may raise exception */
    fegetenv(&env1);
    y = y * x; /* fp op, may raise exception */
    fegetenv(&env2);
}
```

feholdexcept

Saves the current floating-point environment and then clears all exception flags.
This function does not affect the rounding direction and is the same as calling:

```
fegetenv(envp);

feclearexcept(FE_ALL_EXCEPT);

#include <fenv.h>

int feholdexcept(fenv_t *envp)
```

envp	fenv_t	Pointer to floating-point environment
------	--------	---------------------------------------

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 15.8 Example of feholdexcept

```
void ie_feholdexcept(void)
{   /* This function signals underflow if its result is
    denormalized, overflow if its result is infinity,
    and inexact always, but hides spurious exceptions
    occurring from internal computations. */

fenv_t local_env;
int c;
/* can be FP_NAN, FP_INFINITE , FP_ZERO,
   FP_NORMAL, or FP_SUBNORMAL */
long double A=4, B=3, result=0;
feclearexcept(FE_ALL_EXCEPT);

feholdexcept(&local_env);
/* save fp environment */
/* INTERNAL COMPUTATION */
result = pow(A,B);
c = fpclassify(result);
/* inquire about result */
feclearexcept(FE_ALL_EXCEPT);
/* hides spurious exceptions */
feraiseexcept(FE_INEXACT);
/* always raise */
if (c==FP_INFINITE)
    feraiseexcept(FE_OVERFLOW);
else if (c==FP_SUBNORMAL)
    feraiseexcept(FE_UNDERFLOW);
/* RESTORE LOCAL ENV W/ CHNGS */
feupdateenv(&local_env);
}
```

fesetenv

The fesetenv function establishes the floating-point environment represented by the object pointed to by envp. This object will have been set by a previous call to the functions fegetenv, feholdexcept or can use `FE_DEFAULT_ENV`.

```
#include <fenv.h>
void fesetenv(const fenv_t *envp);
```

envp	const fenv_t	Pointer to floating-point environment
------	--------------	---------------------------------------

There is no return value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

For an example of fesetenv refer to “[Example of fegetenv](#)” on page 161.

feupdateenv

This is a multi-step function that takes a saved floating-point environment and does the following:

1. Save the current floating-point environment into temporary storage. This value will be used as a mask to determine which signals to raise.
2. Restore the floating-point environment pointed to by the argument envp.
3. Raise signals in newly restored floating-point environment using values saved in step one.

```
#include <fenv.h>

void feupdateenv(const fenv_t *envp);
```

envp	const fenv_t	Constant pointer to floating-point environment
------	--------------	--

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

There is no return value.

For an example of feupdateenv refer to “[Example of feholdexcept](#)” on page 162.

float.h

The `float.h` header file macros specify the “[Floating point number characteristics](#)” on page 165 for `float`, `double` and `long double` types.

Overview of float.h

The `float.h` header file consists of macros that specify the characteristics of floating point number representation for `float`, `double` and `long double` types.

Floating point number characteristics

These macros are listed in the listing “[Floating point characteristics](#)” on page 165

[“Floating point characteristics” on page 165](#) lists the macros defined in `float.h`. Macros beginning with `FLT` apply to the `float` type; `DBL`, the `double` type; and `LDBL`, the `long double` type.

The `FLT_RADIX` macro specifies the radix of exponent representation.

The `FLT_ROUNDS` specifies the rounding mode. Metrowerks C rounds to nearest.

Listing 16.1 Floating point characteristics

Macro	Description
<code>FLT_MANT_DIG</code> , <code>DBL_MANT_DIG</code> , <code>LDBL_MANT_DIG</code>	The number of base <code>FLT_RADIX</code> digits in the significant.
<code>FLT_DIG</code> , <code>DBL_DIG</code> , <code>LDBL_DIG</code>	The decimal digit precision.
<code>FLT_MIN_EXP</code> , <code>DBL_MIN_EXP</code> , <code>LDBL_MIN_EXP</code>	The smallest negative integer exponent that <code>FLT_RADIX</code> can be raised to and still be expressible.

Listing 16.1 Floating point characteristics

FLT_MIN_10_EXP, DBL_MIN_10_EXP, LDBL_MIN_10_EXP	The smallest negative integer exponent that 10 can be raised to and still be expressible.
FLT_MAX_EXP, DBL_MAX_EXP, LDBL_MAX_EXP	The largest positive integer exponent that FLT_RADIX can be raised to and still be expressible.
FLT_MAX_10_EXP, DBL_MAX_10_EXP, LDBL_MAX_10_EXP	The largest positive integer exponent that 10 can be raised to and still be expressible.
FLT_MIN, DBL_MIN, LDBL_MIN	The smallest positive floating point value.
FLT_MAX, DBL_MAX, LDBL_MAX	The largest floating point value.
FLT_EPSILON, DBL_EPSILON, LDBL_EPSILON	The smallest fraction expressible.

FSp_fopen.h

The FSp_fopen.h header defines functions to open a file based on the Macintosh file system.

Overview of FSp_fopen.h

The FSp_fopen.h header file consist of

- [“FSp_fopen” on page 167](#) a Macintosh file opening for fopen
- [“FSRef_fopen” on page 168](#) a Macintosh file opening for fopen when files exist.
- [“FSRefParentAndFilename_fopen” on page 169](#) A Macintosh file opening for existing and non existing files.

FSp_fopen

The function FSp_fopen opens a file with the Macintosh Toolbox FSpec function and return a FILE pointer.

```
#include <Fsp_fopen.h>
FILE * FSp_fopen (ConstFSSpecPtr spec, const char * open_mode);
```

spec	ConstFSSpecPtr	A toolbox file pointer
open_mode	char *	The open mode

Remarks

This function requires the programmer to include the associated FSp_fopen.c source file in their project. It is not included in the MSL C library.

The FSp_fopen facility opens a file with the Macintosh Toolbox FSRefPtr function and return a FILE pointer.

FSp_fopen.h

Overview of FSp_fopen.h

Macintosh only, this function may not be implemented on all Mac OS versions.

See Also

[“fopen” on page 361](#)

FSRef_fopen

Opens an existing file with FSRef and return a FILE pointer.

```
#include <FSp_fopen.h>
FILE * FSRef_fopen (FSRefPtr spec, const char * open_mode);
```

spec	FSRefPtr	An FSRef pointer for an existing file.
open_mode	char *	The open mode

Remarks

This function requires the programmer to include the associated FSp_fopen.c source file in their project as it is not included in the MSL C library. Also, there are three libraries that may need to be weak linked in a non-Carbon target in order to build when you use the new file system APIs.

- TextCommon
- UnicodeConverter
- UTCUtils

The FSRefPtr facility returns a FILE pointer

Macintosh only, this function may not be implemented on all Mac OS versions.

See Also

[“fopen” on page 361](#)

[“FSRefParentAndFilename_fopen” on page 169](#)

FSRefParentAndFilename_fopen

FSRefParentAndFilename_fopen works on both files that already exist as well as nonexistent files which can be created by the call.

```
#include <Fsp_fopen.h>

FILE * FSRefParentAndFilename_fopen(
    const FSRefPtr theParentRef,
    ConstHFSUniStr255Param theName,
    const char *open_mode);
```

theParentRef	FSRefPtr	A Carbon file pointer to the existing parent FSRef
theName	ConstHFSUniStr255Param	The unicode name for the file to open
open_mode	char *	The open mode

Remarks

This function requires the programmer to include the associated FSp_fopen.c source file in their project as it is not included in the MSL C library. Also, there are three libraries that may need to be weak linked in a non-Carbon target in order to build when you use the new file system APIs.

- TextCommon
- UnicodeConverter
- UTCUtils

The `FSRefParentAndFilename_fopen` facility returns a `FILE` pointer. Macintosh only, this function may not be implemented on all Mac OS versions.

See Also

[“fopen” on page 361](#)

[“FSRef fopen” on page 168](#)

FSp_fopen.h

Overview of FSp_fopen.h

inttypes.h

The header `inttypes.h` defines integer type equivalent symbols and keywords.

Overview of inttypes.h

The `inttypes.h` header file consists of functions for manipulation of greatest-width integers and for converting numeric character string to greatest-width integers. It includes various types and macros to support these manipulations. It also includes formatting symbols for formatted input and output functions.

- [“Greatest-Width Integer Types” on page 171](#) a type used for long long division
- [“Greatest-Width Format Specifier Macros” on page 172](#) formatting specifiers for `printf` and `scanf` family functions.
- [“Greatest-Width Integer Functions” on page 174](#) functions for manipulation and conversion

The header `inttype.h` includes several functions for greatest-width integer manipulation and conversions.

- [“imaxabs” on page 174](#) computes the absolute value of a greatest-width integer
- [“imaxdiv” on page 175](#) computes the quotient and remainder
- [“strtoimax” on page 176](#) string to greatest-width integer
- [“strtoumax” on page 177](#) string to greatest-width unsigned integer
- [“wcstoimax” on page 178](#) wide character string to greatest-width integer
- [“wcstoumax” on page 179](#) wide character string to greatest-width unsigned integer

Greatest-Width Integer Types

One type is defined for greatest-width integer types used for long long division manipulation.

imaxdiv_t

A structure type that can store the value returned by the imaxdiv function as described in [“imaxdiv_t structure elements” on page 172](#).

Listing 18.1 imaxdiv_t structure elements

quot	Defined for the appropriate int, long or long long type
rem	Defined for the appropriate int, long or long long type

Greatest-Width FormatSpecifier Macros

The `inttypes.h` header includes object-like macros that expand to a conversion specifier suitable for use with formatted input and output functions.

The table [“Fprintf Greatest-Width Format Specifiers” on page 172](#) described output formatting macros

The table [“Fscanf Greatest-Width Format Specifiers” on page 173](#) describes input formatting macros.

Listing 18.2 Fprintf Greatest-Width Format Specifiers

Macro Substitution	Specifier	Macro Substitution	Specifier	Macro Substitution	Specifier
PRId8	d	PRId16	hd	PRId32	ld
PRId64	lld	PRIdLEAST8	d	PRIdLEAST16	hd
PRIdLEAST32	ld	PRIdLEAST64	lld	PRIdFAST8	d
PRIdFAST16	hd	PRIdFAST32	ld	PRIdFAST64	lld
PRIdMAX	lld	PRIdPTR	lld	PRlli8	i
PRlli16	hi	PRlli32	li	PRlli64	lli
PRlliLEAST8	i	PRlliLEAST16	hi	PRlliLEAST32	li
PRlliLEAST64	lli	PRlliFAST8	i	PRlliFAST16	hi
PRlliFAST32	li	PRlliFAST64	lli	PRlliMAX	lli
PRlliPTR	li	PRlo8	o	PRlo16	ho
PRlo32	lo	PRlo64	llo	PRloLEAST8	o
PRloLEAST16	ho	PRloLEAST32	lo	PRloLEAST64	llo

Listing 18.2 Fprintf Greatest-Width Format Specifiers

Macro Substitution	Specifier	Macro Substitution	Specifier	Macro Substitution	Specifier
PRIoFAST8	o	PRIoFAST16	ho	PRIoFAST32	lo
PRIoFAST64	llo	PRIoMAX	llo	PRIoPTR	lo
PRIu8	u	PRIu16	hu	PRIu32	lu
PRIu64	llu	PRIuLEAST8	u	PRIuLEAST16	hu
PRIuLEAST32	lu	PRIuLEAST64	llu	PRIuFAST8	u
PRIuFAST16	hu	PRIuFAST32	lu	PRIuFAST64	llu
PRIuMAX	llu	PRIuPTR	lu	PRIx8	x
PRIx16	hx	PRIx32	lx	PRIx64	llx
PRIxLEAST8	x	PRIxLEAST16	hx	PRIxLEAST32	lx
PRIxLEAST64	llx	PRIxFAST8	x	PRIxFAST16	hx
PRIxFAST32	lx	PRIxFAST64	llx	PRIxMAX	llx
PRIxPTR	lx	PRIx8	X	PRIx16	hX
PRIX32	IX	PRIX64	IIX	PRIXLEAST8	X
PRIXLEAST16	hX	PRIXLEAST32	IX	PRIXLEAST64	IIX
PRIXFAST8	X	PRIXFAST16	hX	PRIXFAST32	IX
PRIXFAST64	IIX	PRIXMAX	IIX	PRIXPTR	IX

NOTE

Separate macros are used with input and output functions because different format specifiers are generally required for the `fprintf` and `fscanf` family of functions.

Listing 18.3 Fscanf Greatest-Width Format Specifiers

Macro Substitution	Specifier	Macro Substitution	Specifier	Macro Substitution	Specifier
SCNd8	hhd	SCNd16	hd	SCNd32	ld
SCNd64	lld	SCNdLEAST8	hhd	SCNdLEAST16	hd
SCNdLEAST32	ld	SCNdLEAST64	lld	SCNdFAST8	hhd
SCNdFAST16	hd	SCNdFAST32	ld	SCNdFAST64	lld
SCNdMAX	lld	SCNdPTR	ld	SCNi8	hhi
SCNi16	hi	SCNi32	li	SCNi64	lli

Listing 18.3 Fscanf Greatest-Width Format Specifiers

Macro Substitution	Specifier	Macro Substitution	Specifier	Macro Substitution	Specifier
SCNiLEAST8	hhi	SCNiLEAST16	hi	SCNiLEAST32	li
SCNiLEAST64	lli	SCNiFAST8	hhi	SCNiFAST16	hi
SCNiFAST32	li	SCNiFAST64	lli	SCNiMAX	lli
SCNiPTR	li	SCNo8	hho	SCNo16	ho
SCNo32	lo	SCNo64	llo	SCNoLEAST8	hho
SCNoLEAST16	ho	SCNoLEAST32	lo	SCNoLEAST64	llo
SCNoFAST8	hho	SCNoFAST16	ho	SCNoFAST32	lo
SCNoFAST64	llo	SCNoMAX	llo	SCNoPTR	lo
SCNu8	hhu	SCNu16	hu	SCNu32	lu
SCNu64	llu	SCNuLEAST8	hhu	SCNuLEAST16	hu
SCNuLEAST32	lu	SCNuLEAST64	llu	SCNuFAST8	hhu
SCNuFAST16	hu	SCNuFAST32	lu	SCNuFAST64	llu
SCNuMAX	llu	SCNuPTR	lu	SCNx8	hhx
SCNx16	hx	SCNx32	lx	SCNx64	llx
SCNxLEAST8	hhx	SCNxLEAST16	hx	SCNxLEAST32	lx
SCNxLEAST64	llx	SCNxFAST8	hhx	SCNxFAST16	hx
SCNxFAST32	lx	SCNxFAST64	llx	SCNxMAX	llx
SCNxPTR	lx				

Greatest-Width Integer Functions

The header `inttypes.h` includes several functions for greatest-width integer manipulation and conversions.

imaxabs

Computes the absolute value of a greatest-width integer.

```
#include <inttypes.h>
intmax_t imaxabs(intmax_t j);
```

j	intmax_t	The value being computed
---	----------	--------------------------

Remarks

The behavior is undefined if the result can not be represented.

The imaxabs function returns the absolute value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“abs” on page 461](#)

[“labs” on page 481](#)

imaxdiv

Compute the greatest-width integer quotient and remainder.

```
#include <inttypes.h>
imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);
```

numer	intmax_t	The numerator
denom	intmax_t	The denominator

Remarks

The result is undefined behavior when either part of the result cannot be represented.

The imaxdiv function returns a structure of type [“imaxdiv_t” on page 172](#) storing both the quotient and the remainder values.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“div” on page 476](#)

[“ldiv” on page 482](#)

strtoimax

Character array conversion to a greatest-width integral value.

```
#include <inttypes.h>
intmax_t strtointmax(const char * restrict nptr,
                      char ** restrict endptr, int base);
```

nptr	const char *	A character array to convert
endptr	char **	A pointer to a position in nptr that is not convertible.
base	int	A numeric base between 2 and 36

Remarks

The `strtointmax()` function converts a character array, pointed to by `nptr`, that is expected to represent an integer expressed with the radix `base` to an integer value of type `UINTMAX_MIN`, or `UINTMAX_MAX`. A plus or minus sign (+ or -) prefixing the number string is optional.

The `base` argument in `strtointmax()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then a `strtol` family function converts the character array based on its format. Character arrays beginning with '0' are assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to a long int value.

This function skips leading white space.

The converted value is returned upon success. If a failure occurs zero is returned. If the value is outside of a representable range, the appropriate `INTMAX_MAX`, or `INTMAX_MIN` value is returned and `ERANGE` is stored in `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strtoimax” on page 177](#)
- [“strtol” on page 497](#)
- [“strtoll” on page 501](#)

strtoumax

Character array conversion to a greatest-width integer value.

```
#include <inttypes.h>

uintmax_t strtoumax(const char * restrict nptr,
                     char ** restrict endptr, int base);
```

nptr	const char *	A character array to convert
endptr	char **	A pointer to a position in nptr that is not convertible.
base	int	A numeric base between 2 and 36

Remarks

The `strtoumax()` function converts a character array, pointed to by `nptr`, to an integer value of type `UINTMAX_MIN`, or `UINTMAX_MAX`, in `base`. A plus or minus sign prefix is ignored.

The `base` argument in `strtoumax()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then a `strtoul` family function converts the character array based on its format. Character arrays beginning with '0' are assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to the functions' respective types.

This function skips leading white space.

The converted value is returned upon success. If a failure occurs zero is returned. If the value is outside of a representable range, the appropriate `INTMAX_MIN`, or `UINTMAX_MAX` value is returned and `ERANGE` is stored in `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“strtoimax” on page 176](#)

[“strtoul” on page 502](#)

[“strtoull” on page 503](#)

wcstoimax

Wide character array conversion to a greatest-width integral value.

```
#include <inttypes.h>

intmax_t wcstoimax(const wchar_t * restrict nptr,
                    wchar_t ** restrict endptr, int base);
```

<code>nptr</code>	<code>const wchar_t *</code>	A wide character array to convert
<code>endptr</code>	<code>wchar_t **</code>	A pointer to a position in <code>nptr</code> that is not convertible.
<code>base</code>	<code>int</code>	A numeric base between 2 and 36

Remarks

The `wcstoimax()` function converts a wide character array, pointed to by `nptr`, expected to represent an integer expressed in radix `base` to an integer value of type `INTMAX_MIN`, or `INTMAX_MAX`. A plus or minus sign (+ or -) prefixing the number string is optional.

The base argument in `wcstoimax()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than base are permitted. If base is 0, then a `wcstoll` family function converts the wide character array based on its format. Numerical strings beginning with '0' are assumed to be octal, numerical strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other numerical strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to a long int value.

This function skips leading white space.

The converted value is returned upon success. If a failure occurs zero is returned. If the value is outside of a representable range, the appropriate `INTMAX_MAX`, or `INTMAX_MIN` value is returned and `ERANGE` is stored in `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wcstoumax” on page 179](#)

wcstoumax

Wide character array conversion to a greatest-width integer value.

```
#include <inttypes.h>
uintmax_t wcstoumax(const wchar_t * restrict nptr, wchar_t **
    restrict endptr, int base);
```

<code>nptr</code>	<code>const wchar_t *</code>	A wide character array to convert
<code>endptr</code>	<code>wchar_t **</code>	A pointer to a position in <code>nptr</code> that is not convertible.
<code>base</code>	<code>int</code>	A numeric base between 2 and 36

Remarks

The `wcstoumax()` function converts a wide character array, pointed to by `nptr`, to an integer value of type `UINTMAX_MIN`, or `UINTMAX_MAX`, in base. A plus or minus sign prefix is ignored.

The `base` argument in `wcstoumax()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then and a `wcstoul` family function converts the wide character array based on its format. Numerical strings beginning with '0' are assumed to be octal, numerical strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other numerical strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to the functions' respective types.

This function skips leading white space.

The converted value is returned upon success. If a failure occurs zero is returned. If the value is outside of a representable range, the appropriate `UINTMAX_MIN`, or `UINTMAX_MAX` value is returned and `ERANGE` is stored in `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

["wcstoiimax"](#) on page 178

io.h

The header io.h defines several Windows console functions.

Overview of io.h

The io.h header file consists of

- [“_findclose” on page 182](#) closes a directory search
- [“_findfirst” on page 183](#) opens a directory search
- [“_findnext” on page 183](#) searches a directory
- [“_setmode” on page 184](#) sets the translation for unformatted input and output

NOTE

If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

_finddata_t

The structure `_finddata_t` is defined to store directory information. This structure is used in the directory searching functions.

Listing 19.1 Structure _finddata_t

```
struct _finddata_t {  
    unsigned         attrib;  
    __std(time_t)   time_create;      /* -1 for FAT file systems */  
    __std(time_t)   time_access;     /* -1 for FAT file systems */  
    __std(time_t)   time_write;  
    _fsize_t        size;  
    char            name[260];  
}
```

_findclose

Closes the handle opened by `_findfirst`.

```
#include <io.h>  
int _findclose(long fhandle);
```

fhandle	long	The size in bytes of the allocation
---------	------	-------------------------------------

Remarks

The `fhandle` is returned by `_findfirst`.

Zero is returned on success and a negative one on failure.

Windows compatible header yet may also be implemented in other headers. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a compatibility list.

See Also

[“_findfirst” on page 183](#)

[“_findnext” on page 183](#)

_findfirst

Searches a directory folder.

```
#include <io.h>

long _findfirst(const char *pathname,
                struct _finddata_t * fdata);
```

pathname	const char *	The pathname of the file to be found
fdata	_finddata_t *	A pointer to a file directory information structure

Remarks

Wildcards are allowed in the directory search.

A file handle is returned.

Windows compatible header yet may also be implemented in other headers. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a compatibility list.

See Also

[“_finddata_t” on page 181](#)

[“_findnext” on page 183](#)

[“_findclose” on page 182](#)

_findnext

Continues a directory search began with `_findfirst`.

```
#include <io.h>

int _findnext(long fhandle, struct _finddata_t * fdata);
```

fhandle	long	The handle returned from calling _findfirst
fdata	_finddata_t *	A pointer to a file directory information structure

Remarks

Wildcards are allowed in the directory search.

Zero is returned if the file is found, if no file is found then a negative one is returned.

Windows compatible header yet may also be implemented in other headers. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a compatibility list.

See Also

[“_finddata_t” on page 181](#)

[“_findfirst” on page 183](#)

[“_findclose” on page 182](#)

_setmode

The _setmode function sets the translation mode of the file given by handle

```
#include <io.h>
int _setmode(int handle, int mode);
```

int	handle *	handle
int	mode *	The translation mode

Remarks

The mode must be one of two manifest constants, _O_TEXT or _O_BINARY.

The _O_TEXT mode sets text (a translated) mode. Carriage return-linefeed (CR-LF) combinations are translated into a single linefeed character on input. Linefeed characters are translated into CR-LF combinations on output.

The `_O_BINARY` mode sets `binary` (an untranslated) mode, in which linefeed translations are suppressed.

While `_setmode` is typically used to modify the default translation mode of `stdin` and `stdout`, you can use it on any file. If you apply `_setmode` to the `file` handle for a stream, call `_setmode` before performing any input or output operations on the stream.

The previous translation mode is returned or negative one on failure.

Windows compatible header yet may also be implemented in other headers. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a compatibility list.

io.h

Overview of io.h

iso646.h

The header `isog46.h` defines keyword alternates for the C operator symbols.

Overview of iso646.h

The iso646.h header file consists of equivalent “words” for standard C operators as in [“Operator Keyword Equivalents” on page 187](#).

Listing 20.1 Operator Keyword Equivalents

operator	Keyword Equivalent
<code>&&</code>	<code>and</code>
<code>&=</code>	<code>and_eq</code>
<code>&</code>	<code>bitand</code>
<code> </code>	<code>bitor</code>
<code>~</code>	<code>compl</code>
<code>!=</code>	<code>not_eq</code>
<code> </code>	<code>or</code>
<code>!=</code>	<code>or_eq</code>
<code>^</code>	<code>xor</code>
<code>^=</code>	<code>xor_eq</code>

iso646.h

Overview of iso646.h

limits.h

The `limits.h` header file macros describe the maximum and minimum integral type limits.

Overview of limits.h

The header `limits.h` consists of macros listed in

- [“Integral type limits” on page 189.](#)

Integral type limits

The `limits.h` header file macros describe the maximum and minimum values of integral types.

[“Integral limits” on page 189](#) describes the macros.

Listing 21.1 Integral limits

Macro	Description
<code>CHAR_BIT</code>	Number of bits of smallest object that is not a bit field.
<code>CHAR_MAX</code>	Maximum value for an object of type <code>char</code> .
<code>CHAR_MIN</code>	Minimum value for an object of type <code>char</code> .
<code>SCHAR_MAX</code>	Maximum value for an object of type <code>signed char</code> .
<code>SCHAR_MIN</code>	Minimum value for an object of type <code>signed char</code> .
<code>UCHAR_MAX</code>	Maximum value for an object of type <code>unsigned char</code> .
<code>SHRT_MAX</code>	Maximum value for an object of type <code>short int</code> .
<code>SHRT_MIN</code>	Minimum value for an object of type <code>short int</code> .

Listing 21.1 Integral limits

USHRT_MAX	Maximum value for an object of type <code>unsigned short int</code> .
INT_MAX	Maximum value for an object of type <code>int</code> .
INT_MIN	Minimum value for an object of type <code>int</code> .
LONG_MAX	Maximum value for an object of type <code>long int</code> .
LONG_MIN	Minimum value for an object of type <code>long int</code> .
ULONG_MAX	Maximum value for an object of type <code>unsigned long int</code>
MB_LEN_MAX	Maximum number of bytes in a multibyte character
LLONG_MIN	minimum value for an object of type <code>long long int</code>
LLONG_MAX	Maximum value for an object of type <code>long long int</code>
ULLONG_MAX	Maximum value for an object of type <code>unsigned long long int</code>

locale.h

The `locale.h` header file provides facilities for handling different character sets and numeric and monetary formats.

Overview of locale.h

The facilities that are used for this manipulation of the [“Locale specification” on page 191](#) are:

- [“lconv structure and contents returned by localeconv\(\)” on page 192](#)
- [“localeconv” on page 192](#) to get the locale
- [“setlocale” on page 193](#) to set the locale

Locale specification

The ANSI C Standard specifies that certain aspects of the C compiler are adaptable to different geographic locales. The `locale.h` header file provides facilities for handling different character sets and numeric and monetary formats. Metrowerks C supports the “C” locale by default and a vendor “implementation.

The `lconv` structure, defined in `locale.h`, specifies numeric and monetary formatting characteristics for converting numeric values to character strings. A call to `localeconv()` will return a pointer to an `lconv` structure containing the settings for the “C” locale [Listing 22.1 on page 192](#). An `lconv` member is assigned [“CHAR_MAX” on page 189](#) value if it is not applicable to the current locale.

Listing 22.1 lconv structure and contents returned by localeconv()

```
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;  
    char    *int_curr_symbol;  
    char    int_p_cs_precedes;  
    char    int_n_cs_precedes;  
    char    int_p_sep_by_space;  
    char    int_n_sep_by_space;  
    char    int_p_sign_posn;  
    char    int_n_sign_posn;  
};
```

localeconv

Return the lconv settings for the current locale.

```
#include <locale.h>  
  
struct lconv *localeconv(void);
```

Remarks

localeconv() returns a pointer to an lconv structure for the "C" locale. Refer to Figure 1.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

setlocale

Query or set locale information for the C compiler.

```
#include <locale.h>
char *setlocale( int category, const char *locale);
```

category	int	The part of the C compiler to query or set.
locale	char *	A pointer to the locale

Remarks

The `category` argument specifies the part of the C compiler to query or set.

The argument can have one of six values defined as macros in `locale.h`: `LC_ALL` for all aspects, `LC_COLLATE` for the collating function `strcoll()`, `LC_CTYPE` for `ctype.h` functions and the multibyte conversion functions in `stdlib.h`, `LC_MONETARY` for monetary formatting, `LC_NUMERIC` for numeric formatting, and `LC_TIME` for time and date formatting.

If the `locale` argument is a null pointer, a query is made. The `setlocale()` function returns a pointer to a character string indicating which locale the specified compiler part is set to. The Metrowerks C compiler supports the "C" and "" locale.

Attempting to set a part of the Metrowerks C compiler's locale will have no effect.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

["strcoll" on page 522](#)

locale.h

Overview of locale.h

malloc.h

The header `malloc.h` defines one function, [alloc](#), which lets you allocate memory quickly on from the stack.

Overview of malloc.h

The `malloc.h` header file consists of:

- [“alloc” on page 195](#) that allocates memory from the stack

NOTE If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

alloc

Allocates memory quickly on the stack.

```
#include <malloc.h>  
void *alloc(size_t nbytes);
```

nbytes	size_t	The size in bytes of the allocation
--------	--------	-------------------------------------

Remarks

This function returns a pointer to a block of memory that is `nbytes` long. The block is on the function’s stack. This function works quickly since it decrements

the current stack pointer. When your function exits, it automatically releases the storage.

If you use `alloca()` to allocate a lot of storage, be sure to increase the Stack Size for your project in the Project preferences panel.

If it is successful, `alloca()` returns a pointer to a block of memory. If it encounters an error, `alloca()` returns `NULL`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“calloc” on page 473](#)

[“free” on page 479](#)

[“malloc” on page 484](#)

[“realloc” on page 491](#)

Non Standard <malloc.h> Functions

Various non standard functions are included in the header `malloc.h` for legacy source code and compatibility with operating system frameworks and application programming interfaces

For the function `heapmin` see [“heapmin” on page 107](#) for a full description.

math.h

The `math.h` header file provides floating point mathematical and conversion functions.

Overview of math.h

The header `math.h` includes the following facilities:

Classification Macros

- [“fpclassify” on page 202](#) classifies floating point numbers
- [“isfinite” on page 203](#) tests if a value is a finite number
- [“isnan” on page 203](#) test if a value is a computable number
- [“isnormal” on page 204](#) tests for normal numbers
- [“signbit” on page 204](#) tests for a negative number

Functions

- [“acos” on page 205](#) determines the arccosine
- [“asin” on page 206](#) determines the arcsine
- [“atan” on page 208](#) determines the arctangent
- [“atan2” on page 209](#) determines the arctangent of two variables
- [“ceil” on page 211](#) determines the smallest int not less than x
- [“cos” on page 213](#) determines the cosine
- [“cosh” on page 214](#) determines the hyperbolic cosine
- [“exp” on page 215](#) computes the exponential
- [“fabs” on page 217](#) determines the absolute value
- [“floor” on page 218](#) determines the largest integer not greater than x
- [“fmod” on page 220](#) determines the remainder of a division
- [“frexp” on page 222](#) extracts a value of the mantissa and exponent

-
- “[ldexp](#)” on page 226 computes a value from a mantissa and exponent
 - “[log](#)” on page 228 determines the natural logarithm
 - “[log10](#)” on page 230 determines the logarithm to base 10
 - “[modf](#)” on page 231 separates integer and fractional parts
 - “[pow](#)” on page 233 raises to a power
 - “[sin](#)” on page 235 determines the sine
 - “[sinh](#)” on page 237 determines the hyperbolic sine
 - “[sqrt](#)” on page 238 determines the square root
 - “[tan](#)” on page 240 determines the tangent
 - “[tanh](#)” on page 241 determines the hyperbolic tangent

C9X Implementations

- “[acosh](#)” on page 243 computes the (non-negative) arc hyperbolic cosine
- “[asinh](#)” on page 244 computes the arc hyperbolic sine
- “[atanh](#)” on page 245 computes the arc hyperbolic tangent
- “[cbrt](#)” on page 246 computes the real cube root
- “[copysign](#)” on page 247 gives a value with the magnitude of x and the sign of y
- “[erf](#)” on page 248 computes the error function
- “[erfc](#)” on page 249 complementary error function
- “[exp2](#)” on page 250 computes the base-2 exponential
- “[expm1](#)” on page 251 computes the exponential minus 1
- “[fdim](#)” on page 252 computes the positive difference of its arguments
- “[fma](#)” on page 253 a ternary operation to multiply and add
- “[fmax](#)” on page 254 computes the maximum numeric value of its argument
- “[fmin](#)” on page 255 computes the minimum numeric value of its arguments
- “[gamma](#)” on page 256 computes the gamma function
- “[hypot](#)” on page 257 computes the square root of the sum of the squares of the arguments
- “[isgreater](#)” on page 223 compares two numbers for x greater than y
- “[isgreaterless](#)” on page 224 compares numbers for x not equal to y
- “[isless](#)” on page 224 compares two numbers for x less than y
- “[islessequal](#)” on page 225 compares two numbers for x is less than or equal to y

- “[“isunordered” on page 225](#) compares two numbers for lack of order
- “[“ilogb” on page 258](#) determines the exponent
- “[“lgamma” on page 259](#) computes the log of the absolute value
- “[“log1p” on page 260](#) computes the natural- log of x plus 1
- “[“log2” on page 261](#) computes the base-2 logarithm
- “[“logb” on page 262](#) extracts the exponent of a double value
- “[“modf” on page 263](#) stores floating point information
- “[“nan” on page 264](#) Tests for NaN
- “[“nearbyint” on page 264](#) rounds off the argument to an integral value
- “[“nextafter” on page 265](#) determines the next representable value in the type of the function
- “[“remainder” on page 266](#) computes the remainder x REM y required by IEC 559
- “[“remquo” on page 267](#) computes the same remainder as the remainder function
- “[“rint” on page 268](#) rounds off the argument to an integral value
- “[“rinttol” on page 269](#) rinttol rounds its argument to the nearest long integral value
- “[“round” on page 270](#) rounds its argument to an integral value in floating-point format
- “[“roundtol” on page 271](#) roundtol rounds its argument to the nearest integral value
- “[“scalb” on page 271](#) computes $x * \text{FLT_RADIX}^n$
- “[“trunc” on page 272](#) rounds its argument to an integral value in floating-point format nearest to but no larger than the argument.

Floating point mathematics

The `HUGE_VAL` macro, defined in `math.h`, is returned as an error value by the `strtod()` function. See “[“strtold” on page 500](#) for information on `strtod()`.

Un-optimized x86 `math.h` functions may use the “[“errno” on page 95](#) global variable to indicate an error condition. In particular, many functions set `errno` to `EDOM` (see “[“Error Number Definitions” on page 96](#)”) when an argument is beyond a legal domain.

NaN Not a Number

`NaN` stands for ‘Not a Number’ meaning that it has no relationship with any other number. A `NaN` is neither greater, less, or equal to a number. Whereas infinity is

comparable to a number that is, it is greater than all numbers and negative infinity is less than all numbers.

There are two types of NaN the signalling NaN and quiet NaN. The difference between a signalling NaN and a quiet NaN is that both have a full exponent and both have at least one non-zero significant bit, but the signalling NaN has its 2 most significant bits as 1 whereas a quiet NaN has only the second most significant bit as 1.

Quiet NaN

A quiet NaN is the result of an indeterminate calculation such as zero divided by zero, infinity minus infinity. The IEEE floating-point standard guarantees that a quiet NaN is detectable by requiring that the invalid exception be raised whenever an NaN appears as an operand to any basic arithmetic(+,-,*) or non-arithmetic operation (load/store). Metrowerks Standard Library follows the IEEE specification.

Signaling NaN

A signalling NaN does not occur as a result of arithmetic. A signalling NaN occurs when you load a bad memory value into a floating-point register that happens to have the same bit pattern as a signalling NaN. IEEE 754 requires that in such a situation the invalid exception be raised and the signalling NaN be converted to a quiet NaN so the lifetime of a signalling NaN may be brief.

Floating point error testing.

The math library used for PowerPC Mac OS and Windows (when optimized) is not fully compliant with the 1990 ANSI C standard. One way it deviates is that none of the math functions set errno.

The setting of errno is considered an obsolete mechanism because it is inefficient as well as un-informative. Further more various math facilities may set errno haphazardly for 68k Mac OS.

The MSL math libraries provide better means of error detection. Using fpclassify (which is fully portable) provides a better error reporting mechanism. [“Example usage of error detection” on page 202](#), shows an example code used for error detection that allows you to recover in your algorithm based on the value returned from fpclassify.

Inlined Intrinsics Option

For the Win32 x86 compilers CodeWarrior has an optimization option, “inline intrinsics”. If this option is on the math functions do not set the global variable errno. The debug version of the ANSI C libraries built by Metrowerks has “inline intrinsics” option off and errno is set. The optimized release version of the library has “inline intrinsics” option on, and errno is not set.

Floating Point Classification Macros

Several facilities are available for floating point error classification.

Enumerated Constants

Metrowerks Standard Library includes the following constant types for Floating point evaluation.

`FP_NAN` represents a quiet NaN

`FP_INFINITE` represents a positive or negative infinity

`FP_ZERO` represents a positive or negative zero

`FP_NORMAL` represents all normal numbers

`FP_SUBNORMAL` represents denormal numbers

Remarks

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“NaN Not a Number” on page 199](#)

fpclassify

Classifies floating point numbers.

```
#include <math.h>
int __fpclassify(long double x);
int __fpclassifyd(double x);
int __fpclassifyf(float x);
```

x	float, double or long double	number evaluated
---	------------------------------	------------------

Remarks

An integral value FP_NAN, FP_INFINITE, FP_ZERO, FP_NORMAL and FP_SUBNORMAL.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

- [“isfinite” on page 203](#)
- [“isnan” on page 203](#)
- [“isnormal” on page 204](#)
- [“signbit” on page 204](#)
- [“NaN Not a Number” on page 199](#)

Listing 24.1 Example usage of error detection

```
switch(fpclassify(pow(x,y)))
{
case FP_NAN: // we know y is not an int and <0
case FP_INFINITY: // we know y is an int <0
case FP_NORMAL: // given x=0 we know y=0
case FP_ZERO:// given x<0 we know y >0
}
```

isfinite

The facility `isfinite` tests if a value is a finite number.

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

x	float, double or long double	number evaluated
---	------------------------------	------------------

Remarks

The facility returns true if the value tested is finite otherwise it returns false.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fpclassify” on page 202](#)

isnan

The facility `isnan` test if a value is a computable number.

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

x	float, double or long double	number evaluated
---	------------------------------	------------------

Remarks

This facility is true if the argument is not a number.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fpclassify” on page 202](#)

[“NaN Not a Number” on page 199](#)

isnormal

A test of a normal number.

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

x	float, double or long double	number evaluated
---	------------------------------	------------------

Remarks

This facility is true if the argument is a normal number.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fpclassify” on page 202](#)

signbit

A test for a number that includes a signed bit

```
#include <math.h>
int __signbit(long double x);
int __signbitd(double x);
int __signbit(float x);
```

x	float, double or long double	number evaluated
---	------------------------------	------------------

Remarks

This facility is true if the sign of the argument value is negative.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fpclassify” on page 202](#)

Floating Point Math Facilities

Several facilities are available for floating point manipulations.

acos

This function computes the arc values of cosine, sine, and tangent.

```
#include <math.h>

double acos(double x);
float acosf(float);
long double acosl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

The function `acos()` may set `errno` to `EDOM` if the argument is not in the range of -1 to +1. See [“Floating point error testing.” on page 200](#), for information on newer error testing procedures.

See [“Example of `acos\(\)`, `asin\(\)`, `atan\(\)`, `atan2\(\)` usage.” on page 210](#) for example usage.

The function `acos()` returns the arccosine of the argument `x` in radians. If the argument to `acos()` is not in the range of -1 to +1, the global variable `errno` may be set to `EDOM` and returns 0. See [“Floating point error testing.” on page 200](#), for information on newer error testing procedures.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Inlined Intrinsics Option” on page 201](#)
 - [“cos” on page 213](#)
 - [“errno” on page 95](#)
-

acosf

Implements the acos() function for float type values. See [“acos” on page 205.](#)

acosl

Implements the acos() function for long double type values. See [“acos” on page 205.](#)

asin

Arcsine function.

```
#include <math.h>
double asin(double x);
float asinf(float);
long double asinl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

This function computes the arc values of sine.

The function asin() may set errno to EDOM if the argument is not in the range of -1 to +1. See [“Floating point error testing.” on page 200](#), for information on newer error testing procedures.

The function `asin()` returns the arcsine of `x` in radians. If the argument to `asin()` is not in the range of -1 to +1, the global variable `errno` may be set to `EDOM` and returns 0. See “[Floating point error testing.” on page 200](#), for information on newer error testing procedures.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“sin” on page 235](#)

[“errno” on page 95](#)

See [“Example of acos\(\), asin\(\), atan\(\), atan2\(\) usage.” on page 210](#) for example usage.

asinf

Implements the `asin()` function for float type values. See [“asin” on page 206](#).

asinl

Implements the `asin()` function for long double type values. See [“asin” on page 206](#).

atan

Arctangent function. This function computes the value of the arc tangent of the argument.

```
#include <math.h>
double atan(double x);
float atanf(float);
long double atanl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

The function atan() returns the arc tangent of the argument x in the range [- $\frac{\pi}{2}$, $\frac{\pi}{2}$] radians.

See “[Example of acos\(\), asin\(\), atan\(\), atan2\(\) usage.](#)” on page 210 for example usage.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“tan” on page 240](#)

[“errno” on page 95](#)

atanf

Implements the atan() function for float type values. See [“atan” on page 208](#).

atanl

Implements the atan() function for long double type values. See “[atan](#)” on page [208](#).

atan2

Arctangent function. This function computes the value of the tangent of x/y using the sines of both arguments.

```
#include <math.h>

double atan2(double y, double x);
float atan2f(float, float);
long double atan2l(long double, long double);
```

y	double, float or long double	Value one
x	double, float or long double	Value two

Remarks

A domain error occurs if both x and y are zero.

The function atan2() returns the arc tangent of y/x in the range [- $\frac{\pi}{2}$, + $\frac{\pi}{2}$] radians.

See “[Example of acos\(\), asin\(\), atan\(\), atan2\(\) usage.](#)” on page [210](#) for example usage.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“tan” on page 240](#)

[“errno” on page 95](#)

Listing 24.2 Example of acos(), asin(), atan(), atan2() usage.

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

int main(void)
{
    double x = 0.5, y = -1.0;

    printf("arccos (%f) = %f\n", x, acos(x));
    printf("arcsin (%f) = %f\n", x, asin(x));
    printf("arctan (%f) = %f\n", x, atan(x));
    printf("arctan (%f / %f) = %f\n", y, x, atan2(y, x));

    return 0;
}
```

Output:

```
arccos (0.500000) = 1.047198
arcsin (0.500000) = 0.523599
arctan (0.500000) = 0.463648
arctan (-1.000000 / 0.500000) = -1.107149
```

atan2f

Implements the atan2() function for float type values. See [“atan2” on page 209](#).

atan2l

Implements the atan2() function for long double type values. See [“atan2” on page 209](#).

ceil

Compute the smallest floating point number not less than x .

```
#include <math.h>
double ceil(double x);
float ceilf(float);
long double ceill(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

`ceil()` returns the smallest integer not less than x .

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“floor” on page 218](#)
- [“fmod” on page 220](#)
- [“round” on page 270](#)

Listing 24.3 Example of ceil() usage.

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

int main(void)
{
    double x = 100.001, y = 9.99;

    printf("The ceiling of %f is %f.\n", x, ceil(x));
    printf("The ceiling of %f is %f.\n", y, ceil(y));

    return 0;
}
```

Output:

The ceiling of 100.001000 is 101.000000.
The ceiling of 9.990000 is 10.000000.

ceilf

Implements the ceil() function for float type values. See [“ceil” on page 211.](#)

ceil

Implements the ceil() function for long double type values. See [“ceil” on page 211.](#)

cos

Compute cosine.

```
#include <math.h>

double cos(double x);
float cosf(float);
long double cosl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

`cos()` returns the cosine of `x`. `x` is measured in radians.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“sin” on page 235](#)

[“tan” on page 240](#)

Listing 24.4 Example of cos() usage

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

int main(void)
{
    double x = 0.0;
    printf("The cosine of %f is %f.\n", x, cos(x));

    return 0;
}
```

Output:

The cosine of 0.000000 is 1.000000.

cosf

Implements the cos() function for float type values. See “[cos](#)” on page 213.

cosl

Implements the cos() function for long double type values. See “[cos](#)” on page 213.

cosh

Compute the hyperbolic cosine.

```
double cosh(double x);  
float coshf(float);  
long double coshl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

`cosh()` returns the hyperbolic cosine of `x`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“sinh” on page 237](#)

[“tanh” on page 241](#)

Listing 24.5 cosh() example

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

int main(void)
{
    double x = 0.0;

    printf("Hyperbolic cosine of %f is %f.\n", x, cosh(x));

    return 0;
}
```

Output:

```
Hyperbolic cosine of 0.000000 is 1.000000.
```

coshf

Implements the cosh() function for float type values. See [“cosh” on page 214.](#)

coshl

Implements the cosh() function for long double type values. See [“cosh” on page 214.](#)

exp

Computes the exponent of the function’s argument

```
#include <math.h>

double exp(double x);
float expf(float);
long double expl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

A range error may occur for larger numbers.

`exp()` returns e^x , where e is the natural logarithm base value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“log” on page 228](#)

[“expml” on page 251](#)

[“exp2” on page 250](#)

[“pow” on page 233](#)

Listing 24.6 `exp()` example

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

int main(void)
{
    double x = 4.0;
    printf("The natural logarithm base e raised to the\n");
    printf("power of %f is %f.\n", x, exp(x));

    return 0;
}
```

Output:

The natural logarithm base e raised to the
power of 4.000000 is 54.598150.

expf

Implements the exp() function for float type values. See [“exp” on page 215.](#)

expl

Implements the exp() function for long double type values. See [“exp” on page 215.](#)

fabs

Compute the floating point absolute value.

```
#include <math.h>
double fabs(double x);
float fabsf(float);
long double fabsl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

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

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“floor” on page 218](#)
- [“ceil” on page 211](#)
- [“fmod” on page 220](#)

Listing 24.7 fabs() example

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

int main(void)
{
    double s = -5.0, t = 5.0;
    printf("Absolute value of %f is %f.\n", s, fabs(s));
    printf("Absolute value of %f is %f.\n", t, fabs(t));

    return 0;
}
```

Output:

```
Absolute value of -5.000000 is 5.000000.
Absolute value of 5.000000 is 5.000000.
```

fabsf

Implements the fabs() function for float type values. See [“fabs” on page 217](#).

fabsl

Implements the fabs() function for long double type values. See [“fabs” on page 217](#).

floor

Compute the largest floating point not greater than x .

```
#include <math.h>

double floor(double x);
float floorf(float);
long double floorl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

The function `floor()` returns the largest integer not greater than `x`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“ceil” on page 211](#)
- [“fmod” on page 220](#)
- [“fabs” on page 217](#)

Listing 24.8 floor() example

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

int main(void)
{
    double x = 12.03, y = 10.999;

    printf("Floor value of %f is %f.\n", x, floor(x));
    printf("Floor value of %f is %f.\n", y, floor(y));

    return 0;
}
```

Output:

```
Floor value of 12.030000 is 12.000000.
Floor value of 10.999000 is 10.000000.
```

floorf

Implements the `floor()` function for float type values. See [“floor” on page 218](#).

floorl

Implements the floor() function for long double type values. See “[floor](#)” on page 218.

fmod

Return the floating point remainder of x / y .

```
#include <math.h>
double fmod(double x, double y);
float fmodf(float, float);
long double fmodl(long double, long double);
```

x	double, float or long double	The value to compute
y	double, float or long double	The divider

Remarks

fmod() returns, when possible, the value f such that $x = i y + f$ for some integer i , and $|f| < |y|$. The sign of f matches the sign of x .

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

- [“floor” on page 218](#)
- [“ceil” on page 211](#)
- [“fmod” on page 220](#)
- [“fabs” on page 217](#)

Listing 24.9 Example of fmod() usage.

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

int main(void)
{
    double x = -54.4, y = 10.0;
    printf("Remainder of %f / %f = %f.\n", x, y, fmod(x, y));

    return 0;
}
```

Output:

```
Remainder of -54.400000 / 10.000000 = -4.400000.
```

fmodf

Implements the fmod() function for float type values. See [“fmod” on page 220](#).

fmodl

Implements the fmod() function for long double type values. See [“fmod” on page 220](#).

frexp

Extract the mantissa and exponent. The `frexp()` function extracts the mantissa and exponent of `value` based on the formula $x*2^n$, where the mantissa is $0.5 \leq |x| < 1.0$ and n is an integer exponent.

```
#include <math.h>
double frexp(double value, int *exp);
float frexpf(float, int *);
long double frexpl(long double, int *);
```

<code>x</code>	double, float or long double	The value to compute
<code>exp</code>	int	Exponent

Remarks

`frexp()` returns the double mantissa of `value`. It stores the integer exponent value at the address referenced by `exp`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ldexp” on page 226](#)

[“fmod” on page 220](#)

Listing 24.10 frexp() example

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

int main(void)
{
    double m, value = 12.0;
    int e;

    m = frexp(value, &e);

    printf("%f = %f * 2 to the power of %d.\n", value, m, e);

    return 0;
}

Output:
12.000000 = 0.750000 * 2 to the power of 4.
```

frexpf

Implements the frexp() function for float type values. See [“frexp” on page 222.](#)

frexpl

Implements the frexp() function for long double type values. See [“frexp” on page 222.](#)

isgreater

The facility determine the greater of two doubles. Unlike `x>y` `isgreater` does not raise an invalid exception when `x` and `y` are unordered.

```
#include <math.h>
int isgreater(x, y)
```

x	float, double or long double	number compared
y	float, double or long double	number compared

Remarks

This facility is true if x is greater than y.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

isgreaterless

The facility determines if two numbers are unequal. Unlike `x>y || x<y` `isgreaterless` does not raise an invalid exception when x and y are unordered.

```
#include <math.h>
int isgreaterless(x, y)
```

x	float, double or long double	number compared
y	float, double or long double	number compared

Remarks

This facility returns true if x is greater than or less than y.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

isless

The facility determines the lesser of two numbers.

```
#include <math.h>
int isless(x, y)
```

x	float, double or long double	number compared
y	float, double or long double	number compared

Remarks

This facility is true if x is less than y.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

islessequal

The facility test for less than or equal to comparison. Unlike `x < y || x == y` `islessequal` does not raise an invalid exception when x and y are unordered.

```
#include <math.h>
int islessequal(x, y)
```

x	float, double or long double	number compared
y	float, double or long double	number compared

Remarks

This facility is true if x is less than or equal to y.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

isunordered

The facility compares the order of the arguments.

```
int isunordered(x, y)
```

x	float, double or long double	number compared
y	float, double or long double	number compared

Remarks

This facility is true if the arguments are unordered false otherwise.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

ldexp

Compute a value from a mantissa and exponent. The `ldexp()` function computes $x * 2^{exp}$. This function can be used to construct a double value from the values returned by the `frexp()` function.

```
#include <math.h>

double ldexp(double x, int exp);
float ldexpf(float, int);
long double ldexpl(long double, int);
```

x	double, float or long double	The value to compute
exp	int	Exponent

Remarks

The Function `ldexp()` returns $x * 2^{exp}$.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“frexp” on page 222](#)

[“modf” on page 231](#)

Listing 24.11 Example of ldexp() usage.

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

int main(void)
{
    double value, x = 0.75;
    int e = 4;

    value = ldexp(x, e);

    printf("%f * 2 to the power of %d is %f.\n", x, e, value);

    return 0;
}

Output:
0.750000 * 2 to the power of 4 is 12.000000.
```

ldexpf

Implements the ldexp() function for float type values. See “[ldexp](#)” on page 226.

ldexpl

Implements the ldexp() function for long double type values. See “[ldexp](#)” on page 226.

log

Compute the natural logarithms.

```
#include <math.h>
double log(double x);
float logf(float);
long double logl(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

`log()` returns $\log_e x$. If $x < 0$ the `log()` may assign EDOM to `errno`.

See “[Floating point error testing.” on page 200](#), for information on newer error testing procedures.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“exp” on page 215](#)

[“errno” on page 95](#)

Listing 24.12 log(), log10() example

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

int main(void)
{
    double x = 100.0;

    printf("The natural logarithm of %f is %f\n", x, log(x));
    printf("The base 10 logarithm of %f is %f\n", x, log10(x));

    return 0;
}
```

Output:

```
The natural logarithm of 100.000000 is 4.605170
The base 10 logarithm of 100.000000 is 2.000000
```

logf

Implements the log() function for float type values. See [“log” on page 228.](#)

logl

Implements the log() function for long double type values. See [“log” on page 228.](#)

log10

Compute the base 10 logarithms.

```
#include <math.h>
double log10(double x);
float log10f(float);
long double log10l(long double);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

`log10()` returns $\log_{10}x$. If $x < 0$ `log10()` may assign EDOM to `errno`. See “[Floating point error testing.](#)” on page 200, for information on newer error testing procedures.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“exp” on page 215](#)

[“errno” on page 95](#)

For example of usage refer to [“log\(\), log10\(\) example” on page 229](#)

log10f

Implements the `log10()` function for float type values. See [“log10” on page 230](#).

log10l

Implements the log10() function for long double type values. See “[log10](#)” on [page 230](#).

modf

Separates integer and fractional parts.

```
#include <math.h>

double modf(double value, double *iptr);
float fmodf(float value, float *iptr);
long double modfl(long double value, long double *iptr);
```

value	double, float, or long double	The value to separate
iptr	double, float, or long double	integer part

Remarks

The modf() function separates value into its integer and fractional parts. In other words, modf() separates value such that $\text{value} = f + i$ where $0 \leq f < 1$, and i is the largest integer that is not greater than value.

modf() returns the signed fractional part of value, and stores the integer part in the integer pointed to by iptr.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“frexp” on page 222](#)

[“ldexp” on page 226](#)

Listing 24.13 Example of modf() usage.

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

int main(void)
{
    double i, f, value = 27.04;

    f = modf(value, &i);
    printf("The fractional part of %f is %f.\n", value, f);
    printf("The integer part of %f is %f.\n", value, i);

    return 0;
}
```

Output:

```
The fractional part of 27.040000 is 0.040000.
The integer part of 27.040000 is 27.000000.
```

modff

Implements the modf() function for float type values. See [“modf” on page 231.](#)

modfl

Implements the modf() function for long double type values. See [“modf” on page 231.](#)

pow

Calculate x^y .

```
#include <math.h>

double pow(double x, double y);
float powf(float, float x);
long double powl(long double, long double x);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

The `pow()` function may assign `EDOM` to `errno` if `x` is 0.0 and `y` is less than or equal to zero or if `x` is less than zero and `y` is not an integer.

See “[Floating point error testing.](#)” on page 200, for information on newer error testing procedures.

The function `pow()` returns x^y .

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“sqrt” on page 238](#)

[“Example usage of error detection” on page 202,](#)

Listing 24.14 pow() example

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

int main(void)
{
    double x;

    printf("Powers of 2:\n");
    for (x = 1.0; x <= 10.0; x += 1.0)
        printf("2 to the %4.0f is %4.0f.\n", x, pow(2, x));

    return 0;
}
```

Output:

```
Powers of 2:
2 to the 1 is 2.
2 to the 2 is 4.
2 to the 3 is 8.
2 to the 4 is 16.
2 to the 5 is 32.
2 to the 6 is 64.
2 to the 7 is 128.
2 to the 8 is 256.
2 to the 9 is 512.
2 to the 10 is 1024.
```

powf

Implements the pow() function for float type values. See [“pow” on page 233.](#)

powl

Implements the pow() function for long double type values. See [“pow” on page 233.](#)

sin

Compute sine.

```
#include <math.h>

double sin(double x);
float sinf(float x);
long double sinl(long double x);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

The argument for the sin() function should be in radians. One radian is equal to $360/2\frac{1}{4}$ degrees.

The function sin() returns the sine of x. x is measured in radians.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“cos” on page 213](#)

[“tan” on page 240](#)

Listing 24.15 Example of sin() usage.

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

#define DtoR 2*pi/360

int main(void)
{
    double x = 57.0;
    double xRad = x*DtoR;

    printf("The sine of %.2f degrees is %.4f.\n",x, sin(xRad));

    return 0;
}
```

Output:

The sine of 57.00 degrees is 0.8387.

sinf

Implements the sin() function for float type values. See [“sin” on page 235.](#)

sinl

Implements the sin() function for long double type values. See [“sin” on page 235.](#)

sinh

Compute the hyperbolic sine.

```
#include <math.h>

double sinh(double x);
float sinhf(float x);
long double sinhl(long double x);
```

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Remarks

A range error can occur if the absolute value of the argument is to large.

`sinh()` returns the hyperbolic sine of `x`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“cosh” on page 214](#)

[“tanh” on page 241](#)

Listing 24.16 sinh() example

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

int main(void)
{
    double x = 0.5;
    printf("Hyperbolic sine of %f is %f.\n", x, sinh(x));

    return 0;
}
```

Output:

```
Hyperbolic sine of 0.500000 is 0.521095.
```

sinhf

Implements the sinh() function for float type values. See [“sinh” on page 237](#).

sinhl

Implements the sinh() function for long double type values. See [“sinh” on page 237](#).

sqrt

Calculate the square root.

```
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

x	float, double or long double	value to compute
---	------------------------------	------------------

Remarks

A domain error occurs if the argument is a negative value.

`sqrt()` returns the square root of `x`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“pow” on page 233](#)

Listing 24.17 `sqrt()` example

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

int main(void)
{
    double x = 64.0;

    printf("The square root of %f is %f.\n", x, sqrt(x));

    return 0;
}
```

Output:

The square root of 64.000000 is 8.000000.

sqrtf

Implements the `sqrt()` function for float type values. See [“sqrt” on page 238](#).

sqrtl

Implements the sqrt() function for long double type values. See “[sqrt](#)” on page [238](#).

tan

Computes tangent of the argument.

```
#include <math.h>
double tan(double x);
float tanf(float x);
long double tanl(long double x);
```

x	float, double or long double	value to compute
---	------------------------------	------------------

Remarks

A range error may occur if the argument is close to an odd multiple of pi divided by 2

`tan()` returns the tangent of `x`. `x` is measured in radians.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

[“cos” on page 213](#)

[“sin” on page 235](#)

Listing 24.18 Example of tan() usage.

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

int main(void)
{
    double x = 0.5;

    printf("The tangent of %f is %f.\n", x, tan(x));

    return 0;
}
```

Output:

The tangent of 0.500000 is 0.546302.

tanf

Implements the tan() function for float type values. See [“tan” on page 240.](#)

tanl

Implements the tan() function for long double type values. See [“tan” on page 240.](#)

tanh

Compute the hyperbolic tangent.

```
#include <math.h>

double tanh(double x);
float tanhf(float x);
long double tanhl(long double x);
```

x	float, double or long double	value to compute
---	------------------------------	------------------

Remarks

`tanh()` returns the hyperbolic tangent of `x`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“cosh” on page 214](#)

[“sinh” on page 237](#)

Listing 24.19 `tanh()` example

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

int main(void)
{
    double x = 0.5;

    printf("The hyperbolic tangent of %f is %f.\n", x, tanh(x));

    return 0;
}
```

Output:

The hyperbolic tangent of 0.500000 is 0.462117.

tanhf

Implements the `tanh()` function for float type values. See [“tanh” on page 241](#).

tanhl

Implements the tanh() function for long double type values. See “[tanh](#)” on page [241](#).

HUGE_VAL

The largest floating point value with the same sign possible for a function’s return.

```
#include <math.h>  
Varies by CPU
```

Remarks

If the result of a function is too large to be represented as a value by the return type, the function should return HUGE_VAL. It is the largest floating point value with the same sign as the expected return type.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

C99 Implementations

Although not formally accepted by the ANSI/ISO committee these proposed math functions are already implemented on some platforms.

acosh

Acosh computes the (non-negative) arc or inverse hyperbolic cosine of x in the range $[01, +\infty]$.

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

x	double	The value to compute
---	--------	----------------------

Remarks

A domain error occurs for argument x is less than 1 and a range error occurs if x is too large.

The (non-negative) arc hyperbolic cosine of x.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“acos” on page 205](#)

Listing 24.20 Example of acosh() Usage.

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

int main(void)
{
    double a = 3.14;
    printf("The arc hyperbolic cosine of %f is %f.\n\n",
           a,acosh(a));
    return 0;
}
```

Output:

The arc hyperbolic cosine of 3.140000 is 1.810991.

asinh

Asinh computes the arc or inverse hyperbolic sine.

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

x	double	The value to compute
---	--------	----------------------

Remarks

A range error occurs if the magnitude of x is too large.

The arc hyperbolic sine of the argument x.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“asin” on page 206](#)

Listing 24.21 Example of asinh() Usage

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

int main(void)
{
    double b = 3.14;
    printf("The arc hyperbolic sine of %f is %f.\n\n",
           b, asinh(b));
    return 0;
}
```

Output:

The arc hyperbolic sine of 3.140000 is 1.861813

atanh

The function atanh computes the arc hyperbolic tangent.

```
#include <math.h>

double atanh ( double x );
```

x	double	The value to compute
---	--------	----------------------

Remarks

The arc hyperbolic tangent of x.

A domain error occurs for arguments not in the range [-1,+1]

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“atan” on page 208](#)

Listing 24.22 Example of atanh() Usage

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

int main(void)
{
    double c = 0.5;
    printf("The arc hyperbolic tan of %f is %f.\n\n",
           c, atanh(c));
    return 0;
}
```

Output:

The arc hyperbolic tan of 0.500000 is 0.549306.

cbrt

The cbrt functions compute the real cube root

```
#include <math.h>

double cbrt(double x);
float cbrtf(float fx);
long double cbrtl(long double lx);
```

x	double	The value being cubed
fx	float	The value being cubed
lx	long double	The value being cubed

Remarks

The cbrt functions returns the real cube root of the argument.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“sqrt” on page 238](#)

copysign

The function copysign produces a value with the magnitude of x and the sign of y . The copysign function regards the sign of zero as positive. It produces a NaN with the sign of y if x is NaN.

```
#include <math.h>  
  
double copysign ( double x, double y );
```

x	double	Magnitude
y	double	The sign argument

Remarks

A value with the magnitude of x and the sign of y .

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 24.23 Example of copysign() Usage

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

int main(void)
{
    double e = +10.0;
    double f = -3.0;
    printf("Copysign(%f, %f) = %f.\n\n",
           e,f,copysign(e,f));
    return 0;
}
```

Output:

```
Copysign(10.000000, -3.000000) = -10.000000.
```

erf

The erf function is used in probability.

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

x	double	The value to be computed
---	--------	--------------------------

Remarks

The function erf() is defined as:

```
erf (x) = ( 2/sqrt(pi) *  
( integral from 0 to x of exp(-t^2)dt ))
```

The error function of x is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 24.24 Example of erf() Usage

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

int main(void)
{
    double g = +10.0;
    printf("The error function of (%f) = %f.\n\n",
           g,erf(g));
    return 0;
}
```

Output:

```
The error function of (10.000000) = 1.000000
```

erfc

The erfc() function computes the complement of the error function.

```
#include <math.h>

double erfc ( double x );
```

x	double	The value to be computed
---	--------	--------------------------

The erfc() function computes the complement of the error function of x:

```
erfc(x) = 1 - ( erf (x) = 2/sqrt(pi) *  
( integral from 0 to x of exp(-t^2) dt ))
```

The complementary error function of x is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 24.25 Example of erfc() Usage

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

int main(void)
{
    double h = +10.0;
    printf("The inverse error function of (%f) = %f.\n\n",
           h,erfc(h));
    return 0;
```

Output:

```
The inverse error functions of (10.000000) = 0.000000}
```

exp2

The function exp2 computes the base-2 exponential.

```
#include <math.h>

double exp2 ( double x );
```

x	double	The value to compute
---	--------	----------------------

Remarks

The function exp2 computes the base-2 exponential. In other words $\text{exp2}(b)$ solves for the x in $(b = 2^x)$.

A range error occurs if the magnitude of x is too large

The function returns the base-2 exponential of x : 2^x

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“pow” on page 233](#)

Listing 24.26 Example of exp2() Usage

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

int main(void)
{
    double i = 12;
    printf("2^%f = %f.\n\n", i, i, exp2(i));
    return 0;
}
```

Output:
2^(12.000000) = 4096.000000.

expm1

The function `expm1` computes the base-e exponential minus 1.

```
#include <math.h>

double expm1 ( double x );
```

x	double	The value to compute
---	--------	----------------------

Remarks

The function `expm1` computes the base-e exponential minus 1. Written as:

$$\text{expm1}(x) = (\text{ex}) - 1.0$$

A range error occurs if x is too large. For small magnitude x, `expm1(x)` is expected to be more accurate than $\text{exp}(x) - 1$

The base-e exponential of x, minus 1: $(e^x) - 1$ is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 24.27 Example of expm1() Usage

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

int main(void)
{
    double j = 12;
    printf("(e - 1)^%f = %f.\n\n", j, expm1(j));
    return 0;
}
```

Output:

```
(e - 1)^12.000000 = 162753.791419.
```

fdim

The function fdim computes the positive difference of its arguments

```
#include <math.h>

double fdim ( double x, double y );
```

x	double	Value one
y	double	Value two

Remarks

This function returns the value of $x - y$ if x is greater than y else zero. If x is less than or equal to y a range error may occur

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

Listing 24.28 Example of fdim() Usage

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

int main(void)
{
    double k = 12;
    double l = 4;
    printf("|( %f - %f )| = %f.\n\n", k, l, fdim(k, l));
    return 0;
}
```

Output:

```
| ( 12.000000 - 4.000000 ) | = 8.000000.
```

fma

The fma functions return the sum of the third argument plus the product of the first two arguments rounded as one ternary operation.

```
#include <math.h>

double fma(double x, double y, double z);
float fmaf(float fx, float fy, float fz);
long double fmal(long double lx, long double ly, long double lz);
```

fx	float	An argument to be multiplied
float	The second argument being multiplied	
fz	float	The argument to be added
x	double	An argument to be multiplied
y	double	The second argument being multiplied
z	double	The argument to be added
lx	long double	An argument to be multiplied
ly	long double	The second argument being multiplied
lz	long double	The argument to be added

Remarks

The fma functions compute $(x * y) + z$, rounded as one ternary operation:

The fma functions compute the value (as if) to infinite precision and round once to the result format, according to the rounding mode characterized by the value of FLT_ROUNDS.

The fma functions returns the result of $(x * y) + z$.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“round” on page 270](#)

fmax

The function fmax computes the maximum numeric value of its argument

```
#include <math.h>
double fmax ( double x, double y );
```

x	double	First argument
y	double	Second argument

Remarks

The maximum value of x or y is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fmin” on page 255](#)

Listing 24.29 Example of fmax() Usage

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

int main(void)
{
    double m = 4;
    double n = 6;
    printf("fmax(%f, %f)=%f.\n\n", m, n, fmax(m, n));
    return 0;
}
```

Output:
fmax(4.000000, 6.000000) = 6.000000.

fmin

The function fmin computes the minimum numeric value of its arguments.

```
#include <math.h>

double fmin ( double x, double y );
```

x	double	First argument
y	double	Second argument

Remarks

Fmin returns the minimum numeric value of its arguments

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fmax” on page 254](#)

Listing 24.30 Example of fmin() Usage

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

int main(void)
{
    double o = 4;
    double p = 6;
    printf("fmin(%f, %f) = %f.\n\n", o,p,fmin(o,p));
    return 0;
}
```

Output:

fmin(4.000000, 6.000000) = 4.000000.

gamma

The `gamma()` function computes $\log_e G(x)$.

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

x	double	The value to be computed
---	--------	--------------------------

Remarks

The `gamma()` function computes $\log_e G(x)$, where $G(x)$ is defined as the integral of $(e^{-t}) * t^{(x-1)}$ dt from 0 to infinity. The sign of $G(x)$ is returned in the external integer `signgam`.

The argument x need not be a non-positive integer, ($G(x)$ is defined over the real numbers, except the non-positive integers).

An application wishing to check for error situations should set `errno` to 0 before calling `lgamma()`. If `errno` is non-zero on return,

- If the return value is NaN, an error has occurred.
- A domain error occurs if x is equal to zero or if x is a negative integer.
- A range error may occur.

The gamma function of x is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“lgamma” on page 259](#)

hypot

The function hypot computes the square root of the sum of the squares of the arguments.

```
#include <math.h>
double hypot ( double x, double y );
```

x	double	The first value to be squared
y	double	The second value to be squared

Remarks

Hypot computes the square root of the sum of the squares of x and y without undue overflow or underflow.

A range error may occur.

The square root of the sum of the squares of x and y is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

Listing 24.31 Example of hypot() Usage

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

int main(void)
{
    double r = 4;
    double s = 4;
    printf("(%f^2 + %f^2)^(.5) = %f\n\n", r, s, hypot(r, s));
    return 0;
}
```

Output:

(4.000000^2 + 4.000000^2)^(.5) = 5.656854

ilogb

The `ilogb` functions determine the exponent of the argument as a signed int value.

```
#include <math.h>

int ilogb(double x);
int ilogbf(float y);
int ilogbl(long double z);
```

x	double	The whose exponent is determined
y	float	The whose exponent is determined
z	long double	The whose exponent is determined

Remarks

The `ilogb` functions extract the exponent of x as a signed int value. If x is zero they compute the value `FP_ILOGB0`; if x is infinite they compute the value

INT_MAX; if x is a NaN they compute the value FP_ILOGBNAN; otherwise, they are equivalent to calling the corresponding logb function and casting the returned value to type int.

A rangeerror may occur if x is 0.

The ilogb functions return the exponent of x as a signed int value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“exp” on page 215](#)

Igamma

The lgamma() function computes the same thing as the gamma().

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

x	double	The value to be computed
---	--------	--------------------------

Remarks

The lgamma() function computes the same thing as the gamma() with the addition of absolute value signs $\log|G(x)|$, where $G(x)$ is defined as the integral of $(e^{-t} * t^{(x-1)})dt$ from 0 to infinity.

The sign of $G(x)$ is returned in the external integer signgam. The argument x need not be a non-positive integer, ($G(x)$ is defined over the real numbers, except the non-positive integers).

An application wishing to check for error situations should set errno to 0 before calling lgamma(). If errno is non-zero on return, or the return value is NaN, an error has occurred.

lgamma() may create a range error occurs if x is too large

The log of the absolute value of gamma of x.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“gamma” on page 256](#)

log1p

The function log1p computes the base-e logarithm.

```
#include <math.h>
double log1p ( double x );
```

x	double	The value being computed
---	--------	--------------------------

Remarks

The function log1p computes the base-e logarithm. Which can be denoted as

$$\text{log1p} = \text{loge}(1.0 + x)$$

The value of x must be greater than -1.0.

For small magnitude x, log1p(x) is expected to be more accurate than log(x+1)

- A domain error occurs if x is less than negative one.
- A range error may occur if x is equal to one.

The base-e logarithm of 1 plus x is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“log” on page 228](#)

Listing 24.32 Example of log1p() Usage

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

int main(void)
{
    double u = 5;
    printf("log1p computes log1p(%f) = %f\n\n",u,log1p(u));
    return 0;
}
```

Output:

```
log1p computes log1p(5.000000) = 1.791759
```

log2

The function `log2` computes the base-2 logarithm.

```
#include <math.h>

double log2 ( double x );
```

x	double	The value being computed
---	--------	--------------------------

Remarks

The function `log2` computes the base-2 logarithm which can be denoted as:

$\log_2(x) = \log_2(x)$

The value of x must be positive.

- A domain error may occur if x is less than zero.
- A range error may occur if x is equal to zero.

The base-2 logarithm of x is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“log” on page 228](#)

Listing 24.33 Example of log2() Usage

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

int main(void)
{
    double v = 5;
    printf("log2(%f) = %f\n\n", v, log2(v));
    return 0;
}
```

Output:

```
log2(5.000000) = 2.321928
```

logb

The `logb()` function computes the exponent of `x`.

```
#include <math.h>

double logb ( double x );
```

<code>x</code>	double	The value being computed
----------------	--------	--------------------------

Remarks

The `logb()` function computes the exponent of `x`, which is the integral part of $\log_r |x|$, as a signed floating point value, for non-zero `x`, where `r` is the radix of the machine's floating-point arithmetic. If `x` is subnormal it is treated as though it were normalized.

A range error may occur if `x` is equal to zero.

The exponent of `x` as a signed integral value in the format of the `x` argument.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Inlined Intrinsics Option” on page 201](#)

Listing 24.34 Example of logb() Usage

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

int main(void)
{
    double w = 5;
    printf("logb(%f) = %f\n\n", w, logb(w));
    return 0;
}
```

Output:

```
logb(5.000000) = 2.000000
```

modf

Stores information about a floating point value in a pointer object.

```
#include <math.h>

double modf(double value, double *iptr);
```

value	double	
iptr	double *	object to store information in

Remarks

When the first argument is signed in modf a result with the same sign as that argument is returned.

When the first argument is infinity then modf returns and stores infinity in the object pointed to by iptr.

When the first argument is NaN then modf stores Nan in the object pointed to by iptr and returns NaN.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fmodf” on page 221](#)

nan

The function nan tests for NaN.

```
#include <math.h>
double nan( const char *tagp );
```

tagp	const char *	A character string
------	--------------	--------------------

Remarks

See [“Quiet NaN” on page 200](#), fore more information.

A quiet NAN is returned, if available.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isnan” on page 203](#)

[“NaN Not a Number” on page 199](#)

nearbyint

The function nearbyint rounds off the argument to an integral value.

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

x	double	The value to be computed
---	--------	--------------------------

Remarks

Nearbyint, computes like rint but doesn't raise an inexact exception.

The argument is returned as an integral value in floating point format.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 24.35 Example of nearbyint() Usage

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

int main(void)
{
    double x = 5.7;
    printf("nearbyint(%f) = %f\n\n", x, nearbyint(x));
    return 0;
}
```

Output:
nearbyint(5.700000) = 6.000000

nextafter

The facility `nextafter` determines the next representable value in the type of the function.

```
#include <math.h>

#define nextafter(x,y)
    ( (sizeof(x) == sizeof(float)) ? nextafterf(x,y) :
    (sizeof(x) == sizeof(double)) ? nextafterd(x,y)
```

x	float double long double	current representable value
y	float double long double	direction

Remarks

The facility `nextafter` determines the next representable value in the type of the function, after `x` in the direction of `y`, where `x` and `y` are first converted to the type of the function. Thus, if `y` is less than `x`, `nextafter()` returns the largest representable floating-point number less than `x`.

The next representable value after `x` is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 24.36 Example of nextafter() Usage

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

int main(void)
{
    double y = 7;
    double z = 10;
    printf("nextafter(%f,%f) = %f\n\n",y,z,nextafter(y,z));
    return 0;
}
```

Output:

```
Nextafter(7.000000,10.000000) = 7.000000
```

remainder

Remainder computes the remainder $x \text{ REM } y$ required by IEC 559.

```
#include <math.h>
double remainder ( double x, double y );
```

x	double	The first value
y	double	The second value

Remarks

The `remainder()` function returns the floating point remainder $r = x - ny$ when y is non-zero. The value n is the integral value nearest the exact value x/y . When $|n - x/y| = \frac{1}{2}$, the value n is chosen to be even.

The behavior of `remainder()` is independent of the rounding mode.

The remainder $x \text{ REM } y$ is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“remquo” on page 267](#)

Listing 24.37 Example of remainder() Usage

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

int main(void)
{
    double var1 = 2;
    double var2 = 4;
    printf("remainder(%f,%f) = %f\n\n",var1,var2,remainder(var1,var2));
    return 0;
}
```

Output:

```
remainder(2.000000,4.000000) = 2.000000
```

remquo

The function remquo computes the same remainder as the remainder function.

```
#include <math.h>

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

x	double	First value
y	double	Second value
quo	int*	Pointer to an object quotient

Remarks

The argument quo points to an object whose sign is the sign as x/y and whose magnitude is congruent mod 2^n to the magnitude of the integral quotient of x/y , where $n \geq 3$.

The value of x may be so large in magnitude relative to y that an exact representation of the quotient is not practical.

The remainder of x and y is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“remainder” on page 266](#)

rint

The function rint rounds off the argument to an integral value. Rounds its argument to an integral value in floating-point format using the current rounding direction.

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

x	double	The value to be computed
---	--------	--------------------------

Remarks

The argument in integral value in floating point format is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“rinttol” on page 269](#)

Listing 24.38 Example of rint() Usage

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

int main(void)
{
    double varONE = 2.5;
    printf("rint(%f) = %f\n\n", varONE, rint(varONE));
    return 0;
}

Output:
rint(2.500000) = 2.000000
```

rinttol

Rinttol rounds its argument to the nearest integral value using the current rounding direction.

```
#include <math.h>

long int rinttol ( double x );
```

x	double	Value being rounded
---	--------	---------------------

Remarks

If the rounded range is outside the range of long, result is unspecified

The argument in integral value in floating point format is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“rint” on page 268](#)

round

Round rounds its argument to an integral value in floating-point format.

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

x	double	The value to be rounded
---	--------	-------------------------

Remarks

Rounding halfway cases away from zero, regardless of the current rounding direction.

The argument rounded to an integral value in floating point format nearest to is returned but is never larger in magnitude than the argument.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“roundtol” on page 271](#)

Listing 24.39 Example of round() Usage

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

int main(void)
{
    double varALPHA = 2.4;
    printf("round(%f) = %f\n\n", varALPHA, round(varALPHA));
    return 0;
}
```

Output:
round(2.400000) = 2.000000

roundtol

The function roundtol rounds its argument to the nearest integral value. Rounding halfway cases away from zero, regardless of the current rounding direction.

```
#include <math.h>
long int roundtol ( double round );
```

round	double	The value being rounded
-------	--------	-------------------------

Remarks

If the rounded range is outside the range of long, result is unspecified
The argument rounded to an integral value in long int format is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“round” on page 270](#)

scalb

The function scalb computes $x * \text{FLT_RADIX}^n$ efficiently, not normally by computing FLT_RADIX^n explicitly.

```
#include <math.h>
double scalb ( double x, long int n );
```

x	double	The original value
n	long int	Power value

Remarks

A range error may occur

The function scalb returns `x * FLT_RADIX^n`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

trunc

Trunc rounds its argument to an integral value in floating-point format nearest to but no larger in magnitude than the argument.

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

x	double	The value to be truncated.
---	--------	----------------------------

Remarks

For 68k processors returns an integral value.

The trunc function returns an argument to an integral value in floating-point format.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 24.40 Example of trunc() Usage

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

int main(void)
{
    double varPHI = 2.4108;
    printf("trunc(%f) = %f\n\n", varPHI, trunc(varPHI));
    return 0;
}
```

Output:
trunc(2.410800) = 2.000000

Process.h

The header Process.h defines the threadex functions _beginthreadex and _endthreadex.

Overview of Process.h

The Process.h header file consists of

- “[_beginthread](#)” on page 275 begins a thread
- “[_beginthreadex](#)” on page 276, begins a with local data.
- “[_endthread](#)” on page 277 ends thread for _beginthread
- “[_endthreadex](#)” on page 278, ends thread for _beginthreadex

_beginthread

This function starts a thread for a multi-threaded application.

```
#include <process.h>

long _beginthread (void (__cdecl *inCodeAddress) (void *) thread,
                  unsigned int inStackSize, void *inParameter,
```

thread	void *	The address of the thread function
inStackSize	int	Set by the linkers /STACK switch, 1MB is the default
inParameter	void *	The same as the lpvThreadParameter originally passed, used to pass an initialization routine.

Remarks

The thread runs concurrently with the rest of the code.

The thread handle is returned or zero upon failure.

A Windows only function, this function may not be implemented on all versions.

See Also

[“_endthread” on page 277](#)

[“_beginthreadex” on page 276](#)

_beginthreadex

Begins a thread.

```
#include <process.h>

HANDLE __cdecl _beginthreadex(
    LPSECURITY_ATTRIBUTES inSecurity,
    DWORD inStackSize,
    LPTHREAD_START_ROUTINE inCodeAddress,
    LPVOID inParameter,
    DWORD inCreationFlags,
    LPDWORD inThreadID);
```

inSecurity	LPSECURITY_ATTRIBUTES	Security Attributes, NULL is the default attributes.
inStackSize	DWORD *	Set by the linkers /STACK switch, 1MB is the default
inCodeAddress	LPTHREAD_START_ROUTINE	The address of the function containing the code where the new thread should start.
inParameter	LPVOID	The same as the lpvThreadParameter originally passed, used to pass an initialization routine.

inCreationFlags	DWORD	If zero begins thread immediately, if CREATE_SUSPENDED it waits before executing.
inThreadID	LPDWORD	An variable to store the ID assigned to a new thread.

The function _beginthreadex is similar to the Windows call CreateThread except this function properly creates the local data used by MSL.

A HANDLE variable is returned if successful.

A Windows only function, this function may not be implemented on all versions.

See Also

[“_endthreadex” on page 278](#)

_endthread

This function ends a thread called by _beginthread.

```
#include <process.h>
void _endthread(void);
```

Remarks

This facility has no parameters.

There is no return value for this function.

Windows only compatible function.

See Also

[“_beginthread” on page 275](#)

[“_endthreadex” on page 278](#)

_endthreadex

Exits the thread.

```
#include <process.h>
VOID __cdecl _endthreadex (DWORD inReturnCode);
```

inReturnCode	DWORD	The exit code is passed through this argument.
--------------	-------	--

Remarks

The function _endthreadex is similar to the Windows call ExitThread except this functions properly destroys the thread local data used by MSL.

There is no return.

A Windows only function, this function may not be implemented on all versions.

See Also

[“_beginthreadex” on page 276](#)

setjmp.h

The `setjmp.h` header file provides a means of saving and restoring a processor state. The facilities that do this are:

Overview of `setjmp.h`

The `setjmp.h` header file provides a means of saving and restoring a processor state. The `setjmp.h` functions are typically used for programming error and low-level interrupt handlers.

- The function “[setjmp](#)” on page 281 saves the current calling environment—the current processor state—in its `jmp_buf` argument. The `jmp_buf` type, an array, holds the processor program counter, stack pointer, and relevant data and address registers.
- The function “[longjmp](#)” on page 280 restores the processor to its state at the time of the last `setjmp()` call. In other words, `longjmp()` returns program execution to the last `setjmp()` call if the `setjmp()` and `longjmp()` pair use the same `jmp_buf` variable as arguments.

Non-local jumps and exception handling

Because the `jmp_buf` variable can be global, the `setjmp` and `longjmp` calls do not have to be in the same function body.

A `jmp_buf` variable must be initialized with a call to `setjmp()` before being used with `longjmp()`. Calling `longjmp()` with an un-initialized `jmp_buf` variable may crash the program. Variables assigned to registers through compiler optimization may be corrupted during execution between `setjmp()` and `longjmp()` calls. This situation can be avoided by declaring affected variables as `volatile`.

longjmp

Restore the processor state saved by `setjmp()`.

```
#include <setjmp.h>
void longjmp(jmp_buf env, int val);
```

env	jmp_buf	The current processor state
val	int	A value returned by <code>setjmp()</code>

Remarks

The `longjmp()` function restores the calling environment (i.e. returns program execution) to the state saved by the last called `setjmp()` to use the `env` variable. Program execution continues from the `setjmp()` function. The `val` argument is the value returned by `setjmp()` when the processor state is restored.

The `longjmp` function is redefined when AltiVec support is enabled. The programmer must ensure that both the “to” compilation unit and “from” compilation unit have AltiVec enabled. Failure to do so will create an undesired result.

After `longjmp` is completed, program execution continues as if the corresponding invocation of the `setjmp` macro had just returned the value specified by `val`. The `longjmp` function cannot cause the `setjmp` macro to return the value 0; if `val` is 0, the `setjmp` macro returns the value 1.

The `env` variable must be initialized by a previously executed `setjmp()` before being used by `longjmp()` to avoid undesired results in program execution.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“setjmp” on page 281](#)

[“signal” on page 287](#)

[“abort” on page 459](#)

For example of `longjmp()` usage refer to [“setjmp\(\) example” on page 282](#).

setjmp

Save the processor state for `longjmp()`.

```
#include <setjmp.h>
int setjmp(jmp_buf env);
```

env	jmp_buf	The current processor state
-----	---------	-----------------------------

Remarks

The `setjmp()` function saves the calling environment—data and address registers, the stack pointer, and the program counter—in the `env` argument. The argument must be initialized by `setjmp()` before being passed as an argument to `longjmp()`.

The `setjmp` function is redefined when AltiVec support is enabled. The programmer must ensure that both the “from” compilation unit and “to” compilation unit have AltiVec enabled. Failure to do so will create an undesired result.

When it is first called, `setjmp()` saves the processor state and returns 0. When `longjmp()` is called program execution jumps to the `setjmp()` that saved the processor state in `env`. When activated through a call to `longjmp()`, `setjmp()` returns `longjmp()`'s `val` argument.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“longjmp” on page 280](#)

[“signal” on page 287](#)

[“abort” on page 459](#)

setjmp.h

Overview of setjmp.h

Listing 26.1 setjmp() example

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

// Let main() and doerr() both have
// access to global env
volatile jmp_buf env;

void doerr(void);
int main(void)
{
    int i, j, k;

    printf("Enter 3 integers that total less than 100.\n");
    printf("A zero sum will quit.\n\n");

    // If the total of entered numbers is not less than 100,
    // program execution is restarted from this point.

    if (setjmp(env) != 0)
        printf("Try again, please.\n");

    do {
        scanf("%d %d %d", &i, &j, &k);
        if ( (i + j + k) == 0)
            exit(0);          // quit program
        printf("%d + %d + %d = %d\n\n", i, j, k, i+j+k);
        if ( (i + j + k) >= 100)
            doerr();        // error!
    } while (1);           // loop forever

    return 0;
}

void doerr(void)      // this is the error handler
{
    printf("The total is >= 100!\n");
    longjmp(env, 1);
}
```

Output:

```
Enter 3 integers that total less than 100.
```

A zero sum will quit.

10 20 30
10 + 20 + 30 = 60

-4 5 1000
-4 + 5 + 1000 = 1001

The total is >= 100!

Try again, please.

0 0 0

setjmp.h

Overview of setjmp.h

signal.h

The include file signal.h list the software interrupt specifications.

Overview of signal.h

Signals are software interrupts. There are signals for aborting a program, floating point exceptions, illegal instruction traps, user-sigaled interrupts, segment violation, and program termination.

- These signals, described in [“signal.h Signal descriptions” on page 286](#), are defined as macros in the `signal.h` file.
- [“signal” on page 287](#) specifies how a signal is handled: a signal can be ignored, handled in a default manner, or be handled by a programmer-supplied signal handling function.
- [“raise” on page 290](#) calls the signal handling function
- [“Signal function handling arguments” on page 287](#) describes the pre-defined signal handling macros that expand to functions.

Signal handling

Signals are invoked, or raised, using the `raise()` function. When a signal is raised its associated function is executed.

With the Metrowerks C implementation of `signal.h` a signal can only be invoked through the function [“raise” on page 290](#), and, in the case of the `SIGABRT` signal, through the function [“abort” on page 459](#). When a signal is raised, its signal handling function is executed as a normal function call.

The default signal handler for all signals except `SIGTERM` is `SIG_DFL`. The `SIG_DFL` function aborts a program with the `abort()` function, while the `SIGTERM` signal terminates a program normally with the `exit()` function.

The ANSI C Standard Library specifies that the SIG prefix used by the signal.h macros is reserved for future use. The programmer should avoid using the prefix to prevent conflicts with future specifications of the Standard Library.

The type `typedef char sig_atomic_t` in `signal.h` can be accessed as an incorruptible, atomic entity during an asynchronous interrupt.

The number of signals is defined by `__signal_max` given a value in this header.

WARNING! **Warning:** Using unprotected re-entrant functions such as `printf()`, `getchar()`, `malloc()`, etc. functions from within a signal handler is not recommended in any system that can throw signals in hardware. Signals are in effect interrupts, and can happen anywhere, including when you're already within a function. Even functions that protect themselves from re-entry in a multi-threaded case can fail if you re-enter them from a signal handler.

Listing 27.1 signal.h Signal descriptions

Macro	Description
SIGABRT	Abort signal. This macro is defined as a positive integer value. This signal is called by the <code>abort()</code> function.
SIGBREAK	Terminates calling program.
SIGFPE	Floating point exception signal. This macro is defined as a positive integer value.
SIGILL	Illegal instruction signal. This macro is defined as a positive integer value.
SIGINT	Interactive user interrupt signal. This macro is defined as a positive integer value.
SIGSEGV	Segment violation signal. This macro is defined as a positive integer value.
SIGTERM	Terminal signal. This macro is defined as a positive integer value. When raised this signal terminates the calling program by calling the <code>exit()</code> function.

The `signal()` function specifies how a signal is handled: a signal can be ignored, handled in a default manner, or be handled by a programmer-supplied signal handling function. [“Signal function handling arguments” on page 287](#) describes the pre-defined signal handling macros.

Listing 27.2 Signal function handling arguments

Macro	Description
SIG_IGN	This macro expands to a pointer to a function that returns void. It is used as a function argument in <code>signal()</code> to designate that a signal be ignored.
SIG_DFL	This macro expands to a pointer to a function that returns void. This signal handler quits the program without flushing and closing open streams.
SIG_ERR	A macro defined like <code>SIG_IGN</code> and <code>SIG_DFL</code> as a function pointer. This value is returned when <code>signal()</code> cannot honor a request passed to it.

signal

Set signal handling

```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
```

sig	int	A number associated with the signal handling function
func	void *	A pointer to a signal handling function

Remarks

The `signal()` function returns a pointer to a signal handling routine that takes an `int` value argument.

The `sig` argument is the signal number associated with the signal handling function. The signals defined in `signal.h` are listed in [“signal.h Signal descriptions” on page 286](#).

The `func` argument is the signal handling function. This function is either programmer-supplied or one of the pre-defined signal handlers described in [“Signal function handling arguments” on page 287](#).

When it is raised, a signal handler's execution is preceded by the invocation of `signal(sig, SIG_DFL)`. This call to `signal()` effectively disables the

user's handler. It can be reinstalled by placing a call within the user handler to `signal()` with the user's handler as its function argument.

`signal()` returns a pointer to the signal handling function set by the last call to `signal()` for signal `sig`. If the request cannot be honored, `signal()` returns `SIG_ERR`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“raise” on page 290](#)

[“abort” on page 459](#)

[“atexit” on page 462](#)

[“exit” on page 477](#)

Listing 27.3 Example of signal() usage

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

void userhandler(int);

void userhandler(int sig)
{
    char c;

    printf("userhandler!\nPress return.\n");

    /* wait for the return key to be pressed */
    c = getchar();

}

int main(void)
{
    void (*handlerptr)(int);
    int i;

    handlerptr = signal(SIGINT, userhandler);
    if (handlerptr == SIG_ERR)
        printf("Can't assign signal handler.\n");

    for (i = 0; i < 10; i++) {
        printf("%d\n", i);
        if (i == 5) raise(SIGINT);
    }

    return 0;
}
```

Output:

```
0
1
2
3
4
5
userhandler!
Press return.
```

6

7
8
9

raise

Raise a signal.

```
#include <signal.h>
int raise(int sig);
```

sig	int	A signal handling function
-----	-----	----------------------------

Remarks

The `raise()` function calls the signal handling function associated with signal `sig`.

`raise()` returns a zero if the signal is successful; it returns a nonzero value if it is unsuccessful.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

- [“longjmp” on page 280](#)
- [“signal” on page 287](#)
- [“abort” on page 459](#)
- [“atexit” on page 462](#)
- [“exit” on page 477](#)

For example of `raise()` usage refer to the example for [“Example of signal\(\) usage” on page 289](#)

SIOUX.h

The SIOUX (Simple Input and Output User eXchange) libraries handle Graphical User Interface issues. Such items as menus, windows, and events are handled so your program doesn't need to for C and C++ programs.

Overview of SIOUX

The following section describes the Macintosh versions of the console emulation interface known as SIOUX. The facilities and structure members for the Standard Input Output User eXchange console interface are:

- [“Using SIOUX” on page 291](#) A general description of SIOUX properties.
- [“SIOUX for Macintosh” on page 292](#) the (Simple Input and Output User eXchange) library for the Macintosh Operating Systems.

NOTE If you're porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

Using SIOUX

Sometimes you need to port a program that was originally written for a command line interface such as DOS or UNIX. Or you need to write a new program quickly and don't have the time to write a complete Graphical User Interface that handles windows, menus, and events.

To help you, Metrowerks provides you with the SIOUX libraries, which handles all the Graphical User Interface items such as menus, windows, and titles so your program doesn't need to. It creates a window that's much like a dumb terminal or TTY but with scrolling. You can write to it and read from it with the standard C functions and C++ operators, such as printf(), scanf(), getchar(), putchar() and the C++ inserter and extractor operators << and >>. The SIOUX and WinSIOUX libraries also creates a File menu that lets you save and print the contents of the window. The Macintosh hosted SIOUX includes an Edit menu that lets you cut, copy, and paste the contents in the window. For information on Macintosh redirecting to or from file the stdin, stdout, cout and cin input output or commandline arguments.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

[“Overview of console.h” on page 55.](#)

NOTE	If you're porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.
-------------	---

SIOUX for Macintosh

SIOUX for Macintosh contains the following segments.

- [“Creating a Project with SIOUX” on page 293](#) shows a running SIOUX program.
- [“Customizing SIOUX” on page 295](#) shows how to customize your SIOUX window.
 - [“The SIOUXSettings structure” on page 295](#) list structure members that may be set for altering SIOUX’s appearance
- [“Using SIOUX windows in your own application” on page 300](#) contains information for using Mac OS facilities with in your SIOUX project.
 - [“path2fss” on page 301](#) a function similar to PBMakeFSSpec.
 - [“SIOUXHandleOneEvent” on page 302](#) allows you to use an even in SIOUX
 - [“SIOUXSetTitle” on page 303](#) allows you to specify a custom title for SIOUX’s window

NOTE	A WASTE© by Marco Piovanelli based SIOUX console is available as a pre-release version. This will allow screen output of
-------------	---

over 32k characters. All normal SIOUX functions should work but normal pre-release precautions should be taken. Please read all release notes.

The window is a re-sizeable, scrolling text window, where your program reads and writes text. It saves up to 32K of your program's text.

With the commands from the Edit menu, you can cut and copy text from the SIOUX window and paste text from other applications into the SIOUX window. With the commands in the File menu, you can print or save the contents of the SIOUX window.

To stop your program at any time, press Command-Period or Control-C. The SIOUX application keeps running so you can edit or save the window's contents. If you want to exit when your program is done or avoid the dialog asking whether to save the window, see [“Changing what happens on quit” on page 300](#)

To quit out of the SIOUX application at any time, choose Quit from the File menu. If you haven't saved the contents of the window, the application displays a dialog asking you whether you want to save the contents of the window now. If you want to remove the status line, see [“Showing the status line” on page 300](#).

Creating a Project with SIOUX

To use the SIOUX library, create a project from a project stationery pads that creates an Console style project.

In this chapter, standard input and standard output refer to `stdin`, `stdout`, `cin`, and `cout`. Standard error reporting such as `stderr`, `clog` and `cerr` is not redirected to a file using `ccommand()`.

If you want only to write to or read from standard input and output, you don't need to call any special functions or include any special header files. When your program refers to standard input or output, the SIOUX library kicks in automatically and creates a SIOUX window for it.

NOTE	Remember that functions like <code>printf()</code> and <code>scanf()</code> use standard input and output even though these symbols do not appear in their parameter lists.
-------------	---

If you want to customize the SIOUX environment, you must `#include SIOUX.h` and modify `SIOUXSettings` before you use standard input or output. As soon as you use one of them, SIOUX creates a window and you cannot modify it. For more information, see [“Customizing SIOUX” on page 295](#).

If you want to use a SIOUX window in a program that has its own event loop, you must modify `SIOUXSettings` and call the function `SIOUXHandleOneEvent()`. For more information, see [“Using SIOUX windows in your own application” on page 300.](#)

If you want to add SIOUX to a project you already created, the project must contain certain libraries.

A PPC project must either contain at least these libraries:

- `MSL_All_PPC.Lib`
- `InterfaceLib`
- `MathLib`

Or at least:

- `MSL_C_PPC.Lib`
- `MSL_CPP_PPC.Lib` (for C++)
- `MSL_SIOUX_PPC.Lib`
- `MSL_Runtime_PPC.Lib`
- `InterfaceLib`
- `MathLib`

A Carbon project must either contain at least these libraries:

- `MSL_All_Carbon.Lib`
- `CarbonLib`

Or at least:

- `MSL_C_Carbon.Lib`
- `MSL_CPP_Carbon.Lib` (for C++)
- `MSL_SIOUX_Carbon.Lib`
- `CarbonLib`

A Mach-O project must contain at least these libraries:

- `MSL_All_Mach-O.lib`
- `MSL_SIOUX_Mach-O.lib` (to be implemented)

Or at least:

- `MSL_C_Mach-O.lib`
- `MSL_CPP_Mach-O.lib` (for C++)
- `MSL_SIOUX_Mach-O.lib` (to be implemented)

- MSL_Runtime_Mach-O.lib

Customizing SIOUX

The following sections describe how you can customize the SIOUX environment by modifying the structure `SIOUXSettings`. SIOUX examines the data fields of `SIOUXSettings` to determine how to create the SIOUX window and environment.

NOTE	To customize SIOUX, you must modify <code>SIOUXSettings</code> before you call any function that uses standard input or output. If you modify <code>SIOUXSettings</code> afterwards, SIOUX does not change its window.
-------------	--

The first three sections, “[Changing the font and tabs](#)” on page 297, “[Changing the size and location](#)” on page 299, and “[Showing the status line](#)” on page 300, describe how to customize the SIOUX window. The next section, “[Changing what happens on quit](#)” on page 300, describes how to modify how SIOUX acts when you quit it. The last section, “[Using SIOUX windows in your own application](#)” on page 300, describes how you can use a SIOUX window in your own Macintosh program.

[“The `SIOUXSettings` structure” on page 295](#) summarizes what’s in the `SIOUXSettings` structure.

Listing 28.1 The `SIOUXSettings` structure

This field...	Specifies...
char initializeTB	Whether to initialize the Macintosh toolbox.
char standalone	Whether to use your own event loop or SIOUX’s.
char setupmenus	Whether to create File and Edit menus for the application.
char autocloseonquit	Whether to quit the application automatically when your program is done.
char asktosaveonclose	Query the user whether to save the SIOUX output as a file, when the program is done.
char showstatusline	Whether to draw the status line in the SIOUX window.

Listing 28.1 The SIOUXSettings structure

This field...	Specifies...
short tabspace	If greater than zero, substitute a tab with that number of spaces. If zero, print the tabs.
short column	The number of characters per line that the SIOUX window will contain.
short rows	The number of lines of text that the SIOUX window will contain.
short toppixel	The location of the top of the SIOUX window.
short leftpixel	The location of the left of the SIOUX window.
short fontid	The font in the SIOUX window.
short fontsize	The size of the font in the SIOUX window.
short stubmode	SIOUX acts like a stubs library
char usefloatingwindows	(Carbon) use non floating front window
short fontface	The style of the font in the SIOUX window.
int sleep	The default value for the sleep setting is zero. Zero gets the most speed out of SIOUX by telling the system to not give time to other processes during a WaitNextEvent call. A more appropriate setting (that is more friendly to other processes) is to set the sleep value to GetCaretTime().

[“Example of customizing a SIOUX Window” on page 297](#) contains a small program that customizes a SIOUX window.

Listing 28.2 Example of customizing a SIOUX Window

```
#include <stdio.h>
#include <sioux.h>
#include <MacTypes.h>
#include <Fonts.h>

int main(void)
{
    short familyID;

    /* Don't exit the program after it runs or ask whether
       to save the window when the program exit */
    SIOUXSettings.autocloseonquit = false;
    SIOUXSettings.asktosaveonclose = false;

    /* Don't show the status line */
    SIOUXSettings.showstatusline = false;

    /* Make the window large enough to fit 1 line
       of text that contains 12 characters. */
    SIOUXSettings.columns = 12;
    SIOUXSettings.rows = 1;

    /* Place the window's top left corner at (5,40). */
    SIOUXSettings.toppixel = 40;
    SIOUXSettings.leftpixel = 5;

    /* Set the font to be 48-point, bold, italic Times. */
    SIOUXSettings.fontsize = 48;
    SIOUXSettings.fontface = bold + italic;
    GetFNum("\ptimes", &familyID);
    SIOUXSettings.fontid = familyID;

    printf("Hello World!");

    return 0;
}
```

Changing the font and tabs

This section describes how to change how SIOUX handles tabs with the field `tabspaces` and how to change the font with the fields `fontid`, `fontsize`, and `fontface`.

NOTE	The status line in the SIOUX window writes its messages with the font specified in the fields <code>fontid</code> , <code>fontsize</code> , and <code>fontface</code> . If that font is too large, the status line may be unreadable. You can remove the status line by setting the field <code>showstatusline</code> to <code>false</code> , as described in “Showing the status line” on page 300 .
-------------	---

To change the font in the SIOUX window, set `fontid` to one of these values defined in the header file `FONTS.H`:

- `courier` where the ID is `kFontIDCourier`
- `geneva` where the ID is `kFontIDGeneva`
- `helvetica` where the ID is `kFontIDHelvetica`
- `monaco` where the ID is `kFontIDMonaco`
- `newYork` where the ID is `kFontIDNewYork`
- `symbol` where the ID is `kFontIDSymbol`
- `times` where the ID is `kFontIDTimes`

By default, `fontid` is `monaco`.

To change the character style for the font, set `fontface` to one of these values:

- `normal`
- `bold`
- `italic`
- `underline`
- `outline`
- `shadow`
- `condense`
- `extend`

To combine styles, add them together. For example, to write text that's bold and italic, set `fontface` to `bold + italic`. By default, `fontface` is `normal`.

To change the size of the font, set `fontsize` to the size. By default, `fontsize` is 9.

The field `tabspaces` controls how SIOUX handles tabs. If `tabspaces` is any number greater than 0, SIOUX prints that number of spaces required to get to the next tab position instead of a tab. If `tabspaces` is 0, it prints a tab. In the SIOUX window, a tab looks like a single space, so if you are printing a table, you should set `tabspaces` to an appropriate number, such as 4 or 8. By default, `tabspaces` is 4.

The sample below sets the font to 12-point, bold, italic New York and substitutes 4 spaces for every tab:

```
SIOUXSettings.fontsize = 12;  
SIOUXSettings.fontface = bold + italic;  
SIOUXSettings.fontid = kFontIDNewYork;  
SIOUXSettings.tabspace = 4;
```

Changing the size and location

SIOUX lets you change the size and location of the SIOUX window.

To change the size of the window, set `rows` to the number of lines of text in the window and set `columns` to the number of characters in each line. SIOUX checks the font you specified in `fontid`, `fontsize`, and `fontface` and creates a window that will be large enough to contain the number of lines and characters you specified. If the window is too large to fit on your monitor, SIOUX creates a window only as large as the monitor can contain.

For example, the code below creates a window that contains 10 lines with 40 characters per line:

```
SIOUXSettings.rows = 10;  
SIOUXSettings.columns = 40;
```

By default, the SIOUX window contains 24 rows with 80 characters per row.

To change the position of the SIOUX window, set `toppixel` and `leftpixel` to the point where you want the top left corner of the SIOUX window to be. By setting `toppixel` to 38 and `leftpixel` to 0, you can place the window as far left as possible and just under the menu bar. Notice that if `toppixel` is less than 38, the SIOUX window is under the menu bar. If `toppixel` and `leftpixel` are both 0, SIOUX doesn't place the window at that point but instead centers it on the monitor.

For example, the code below places the window just under the menu bar and near the left edge of the monitor:

```
SIOUXSettings.toppixel = 40;  
SIOUXSettings.leftpixel = 5;
```

Changing what happens on quit

The fields `autocloseonquit` and `asktosaveonclose` let you control what SIOUX does when your program is over and SIOUX closes its window.

The field `autocloseonquit` determines what SIOUX does when your program has finished running. If `autocloseonquit` is `true`, SIOUX automatically exits. If `autocloseonquit` is `false`, SIOUX continues to run, and you must choose `Quit` from the `File` menu to exit. By default, `autocloseonquit` is `false`.

NOTE You can save the contents of the SIOUX window at any time by choosing `Save` from the `File` menu.

The field `asktosaveonclose` determines what SIOUX does when it exits. If `asktosaveonclose` is `true`, SIOUX displays a dialog asking whether you want to save the contents of the SIOUX window. If `asktosaveonclose` is `false`, SIOUX exits without displaying the dialog. By default, `asktosaveonclose` is `true`.

For example, the code below quits the SIOUX application as soon as your program is done and doesn't ask you to save the output:

```
SIOUXSettings.autocloseonquit = true;  
SIOUXSettings.asktosaveonclose = false;
```

Showing the status line

The field `showstatusline` lets you control whether the SIOUX window displays a status line, which contains such information as whether the program is running, handling output, or waiting for input. If `showstatusline` is `true`, the status line is displayed. If `showstatusline` is `false`, the status line is not displayed. By default, `showstatusline` is `false`.

Using SIOUX windows in your own application

This section explains how you can limit how much SIOUX controls your program. But first, you need to understand how SIOUX works with your program. You can consider the SIOUX environment to be an application that calls your `main()` function as just another function. Before SIOUX calls `main()`, it performs some initialization to set up the Macintosh Toolbox and its menu. After `main()` completes, SIOUX cleans up what it created. Even while `main()` is running, SIOUX sneaks in whenever it performs input or output, acting on any menu you've chosen or command key you've pressed.

However, SIOUX lets you choose how much work it does for you. You can choose to handle your own events, set up your own menus, and initialize the Macintosh Toolbox yourself.

When you want to write an application that handles its own events and uses SIOUX windows for easy input and output, set the field `standalone` to `false` before you use standard input or output. SIOUX doesn't use its event loop and sets the field `autocloseonquite` to `true` for you, so the application exits as soon as your program is done. In your event loop, you need to call the function `SIOUXHandleOneEvent()`, described on [“Using SIOUX windows in your own application” on page 300](#).

When you don't want to use SIOUX's menus, set the field `setupmenus` to `false`. If `standalone` is also `false`, you won't be able to create menus, and your program will have none. If `standalone` is `true`, you can create and handle your own menus.

When you want to initialize the Macintosh Toolbox yourself, set the field `initializeTB` to `false`. The field `standalone` does not affect `initializeTB`.

For example, these lines set up SIOUX for an application that handles its own events, creates its own menus, and initializes the Toolbox:

```
SIOUXSettings.standalone = false;  
SIOUXSettings.setupmenus = false;  
SIOUXSettings.initializeTB = false;
```

path2fss

This function is similar to `PBMakeFSSpec`.

```
#include <path2fss.h>  
OSErr __path2fss  
(const char * pathName, FSSpecPtr spec)
```

pathname	const char *	The path name
spec	FSSpecPtr	A file specification pointer

Remarks

This function is similar to `PBMakeFSSpec` with three major differences:

- Takes only a path name as input (as a C string) no parameter block.
- Only makes FSSpecs for files, not directories.
- Works on **any** HFS Mac (Mac 512KE, Mac Plus or later) under any system version that supports HFS.
- Deals correctly with MFS disks (correctly traps file names longer than 63 chars and returns bdNamErr).

Like PBMakFSSpec, this function returns fnfErr if the specified file does not exist but the FSSpec is still valid for the purposes of creating a new file.

Errors are returned for invalid path names or path names that specify directories rather than files.

Macintosh only, This function may not be implemented on all Mac OS versions.

See Also

“Inside Macintosh: Files”

SIOUXHandleOneEvent

Handles an event for a SIOUX window.

```
#include <SIOUX.h>
Boolean SIOUXHandleOneEvent(EventRecord *event);
```

event	EventRecord*	A pointer to a toolbox event
-------	--------------	------------------------------

Remarks

Before you handle an event, call SIOUXHandleOneEvent() so SIOUX can update its windows when necessary. The argument event should be an event that WaitNextEvent() or GetNextEvent() returned. The function returns true if it handled the event and false if it didn't. If event is a NULL pointer, the function polls the event queue until it receives an event.

If it handles the event, SIOUXHandleOneEvent() returns true. Otherwise, SIOUXHandleOneEvent() returns false.

Macintosh only, This function may not be implemented on all Mac OS versions.

Listing 28.3 Example of SIOUXHandleOneEvent() usage.

```
void MyEventLoop(void)
{
    EventRecord event;
    RgnHandle cursorRgn;
    Boolean gotEvent, SIOUXDidEvent;

    cursorRgn = NewRgn();

    do {
        gotEvent = WaitNextEvent(everyEvent, &event,
                                 MyGetSleep(), cursorRgn);

        /* Before handling the event on your own,
         * call SIOUXHandleOneEvent() to see whether
         * the event is for SIOUX.
        */
        if (gotEvent)
            SIOUXDidEvent = SIOUXHandleOneEvent(&event);

        if (!SIOUXDidEvent)
            DoEvent(&event);

    } while (!gDone)
}
```

SIOUXSetTitle

To set the title of the SIOUX output window.

```
include <SIOUX.h>

extern void SIOUXSetTitle(unsigned char title[256])
```

title	unsigned char []	A pascal string
-------	------------------	-----------------

Remarks

You must call the `SIOUXSetTitle()` function after an output to the SIOUX window. The function `SIOUXSetTitle()` does not return an error if the title is

not set. A write to console is not performed until a new line is written, the stream is flushed or the end of the program occurs.

NOTE The argument for `SIOUXSetTitle()` is a pascal string, not a C style string.

There is no return value from `SIOUXSetTitle()`

Macintosh only, This function may not be implemented on all Mac OS versions.

Listing 28.4 Example of `SIOUXSetTitle()` usage.

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

int main(void)
{
    printf("Hello World\n");
    SIOUXSetTitle("\pMy Title");

    return 0;
}
```

stat.h

The header file `stat.h` contains several functions that are useful for porting a program from UNIX.

Overview of stat.h

The facilities in `stat.h` include:

- [“Stat Structure and Definitions.”](#) the `stat` struct and types
- [“chmod” on page 308](#) to change a files attributes
- [“fstat” on page 309](#) to get information on an open file
- [“mkdir” on page 311](#) to make a directory for folder
- [“stat” on page 312](#) to get statistics of a file

stat.h and UNIX Compatibility

Generally, you don’t want to use these functions in new programs. Instead, use their counterparts in the native API.

NOTE	If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.
-------------	---

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

Stat Structure and Definitions

The header stat.h includes the `stat` (and `_stat`, for Windows) structure, listed in “[The stat or _stat Structure” on page 306](#). It also has several type definitions and file mode definitions. The necessary types are listed in “[Defined types” on page 306](#). The file modes are listed in “[File Modes” on page 307](#). File mode macros are listed in “[File Mode Macros Non Windows” on page 307](#) and “[File Mode Macros Windows Only” on page 308](#).

Listing 29.1 Defined types

Type	Used to Store
<code>dev_t</code>	Device type
<code>gid_t</code>	The file size
<code>ino_t</code>	File information
<code>mode_t</code>	File Attributes
<code>nlink_t</code>	The number of links
<code>off_t</code>	The file size in bytes
<code>uid_t</code>	The user's ID

Listing 29.2 The stat or _stat Structure

Type	Variable	Purpose
<code>mode_t</code>	<code>st_mode</code>	File mode, see “ File Modes” on page 307
<code>ino_t</code>	<code>st_ino</code>	File serial number
<code>dev_t</code>	<code>st_dev</code>	ID of device containing this file
<code>dev_t</code>	<code>std_rdev</code> (Windows)	ID of device containing this file
<code>nlink_t</code>	<code>st_nlink</code>	Number of links
<code>uid_t</code>	<code>st_uid</code>	User ID of the file's owner
<code>gid_t</code>	<code>st_gid</code>	Group ID of the file's group
<code>off_t</code>	<code>st_size</code>	File size in bytes

Listing 29.2 The stat or _stat Structure

Type	Variable	Purpose
__std(time_t)	st_atime	Time of last access
__std(time_t)	st_mtime	Time of last data modification
__std(time_t)	st_ctime	Time of last file status change
long	st_blksize	Optimal blocksize
long	st_blocks	Blocks allocated for file

File Modes

File mode information.

Table 29.1 File Modes

File Mode	Purpose
S_IFMT	File type
S_IFIFO	FIFO queue
S_IFCHR	Character special
S_IFDIR	Directory
S_IFBLK	Blocking stream (non Windows)
S_IFREG	Regular
S_IFLNK	Symbolic link (non Windows)
S_IFSOCK	Socket (non Windows)

Listing 29.3 File Mode Macros Non Windows

File Mode	Purpose
S_IRGRP	Read permission file group class
S_IROTH	Read permission file other class
S_IRUSR	Read permission file owner class
S_IWGXG	Permissions for file group class
S_IWXO	Permissions for file other class

Listing 29.3 File Mode Macros Non Windows

File Mode	Purpose
S_IRWXU	Permissions for file owner class
S_ISGID	Set group ID on execution
S_ISUID	Set user ID on execution
S_IWGRP	Write permission file group class
S_IWOTH	Write permission file other class
S_IWUSR	Write permission file owner class
S_IXGRP	Exec permission file group class
S_IXOTH	Exec permission file other class
S_IXUSR	Exec permission file owner class

Listing 29.4 File Mode Macros Windows Only

File Mode	Purpose
S_IEXEC	Execute/search permission, owner (Windows)
S_IREAD	Read permission, owner (Windows Only)
S_IWRITE	Write permission, owner (Windows Only)

chmod

Gets or sets file attributes.

```
#include <stat.h>
int chmod(const char *, mode_t);
int _chmod(const char *, mode_t);
```

path	const char *	The file to change modes on
mod	mode_t	A mask of the new file attributes

Remarks

The file attributes as a mask is returned or a negative one on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fstat” on page 309](#)

fstat

Gets information about an open file.

```
#include <stat.h>
int fstat(int fildes, struct stat *buf);
int _fstat(int fildes, struct stat *buf);
```

fildes	int	A file descriptor
buf	struct stat *	The stat structure address

Remarks

This function gets information on the file associated with `fildes` and puts the information in the structure that `buf` points to. The structure contains the fields listed in [“Stat Structure and Definitions” on page 306](#).

If it is successful, `fstat()` returns zero. If it encounters an error, `fstat()` returns `-1` and sets `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“stat” on page 312](#)

Listing 29.5 Example of fstat() usage.

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

int main(void)
{
    struct stat info;
    int fd;

    fd = open("mytest", O_WRONLY | O_CREAT | O_TRUNC);
    write(fd, "Hello world!\n", 13);

    fstat(fd, &info);
    /* Get information on the open file. */

    printf("File mode:          0x%lX\n", info.st_mode);
    printf("File ID:            0x%lX\n", info.st_ino);
    printf("Volume ref. no.:   0x%lX\n", info.st_dev);
    printf("Number of links:   %hd\n", info.st_nlink);
    printf("User ID:           %lu\n", info.st_uid);
    printf("Group ID:          %lu\n", info.st_gid);
    printf("File size:          %ld\n", info.st_size);
    printf("Access time:        %s", ctime(&info.st_atime));
    printf("Modification time: %s", ctime(&info.st_mtime));
    printf("Creation time:     %s", ctime(&info.st_ctime));

    close(fd);

    return 0;
}
```

This program may print the following:

```
File mode:          0x800
File ID:            0x5ACA
Volume ref. no.:   0xFFFFFFFF
Number of links:   1
User ID:           200
Device type:       0
File size:          13
```

mkdir

Makes a folder.

```
#include <stat.h>

int mkdir(const char *path, int mode);
int _mkdir(const char *path);
```

path	const char *	The path name
mode	int	The open mode (Not applicable for Windows)

Remarks

This function creates the new folder specified in `path`. It ignores the argument `mode`.

If it is successful, `mkdir()` returns zero. If it encounters an error, `mkdir()` returns -1 and sets `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“unlink” on page 602](#)
- [“rmdir” on page 596](#)

stat.h

Overview of stat.h

Listing 29.6 Example for mkdir() on Macintosh OS

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

int main(void)
{
    if( mkdir(":Asok", 0) == 0)
        printf("Folder Asok is created");

    return 0;
}

Windows
#include <stdio.h>
#include <stat.h>

int main(void)
{
    if( mkdir(".\\Asok") == 0)
        printf("Folder Asok is created");

    return 0;
}
```

Output

Creates a folder named Asok as a sub-folder of the current folder

stat

Gets information about a file.

```
#include <stat.h>

int stat(const char *path, struct stat *buf);
int _stat(const char *path, struct stat *buf);
```

path	const char *	The path name
buf	struct stat *	A pointer to the stat struct

Remarks

This function gets information on the file specified in `path` and puts the information in the structure that `buf` points to. The structure contains the fields listed in [“Stat Structure and Definitions” on page 306](#).

If it is successful, `stat()` returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fstat” on page 309](#)

[“uname” on page 620](#)

Listing 29.7 Example of stat() usage.

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

int main(void)
{
    struct stat info;

    stat("Akbar:System Folder:System", &info);
    /* Get information on the System file. */

    printf("File mode:          0x%lX\n", info.st_mode);
    printf("File ID:           0x%lX\n", info.st_ino);
    printf("Volume ref. no.:   0x%lX\n", info.st_dev);
    printf("Number of links:    %hd\n", info.st_nlink);
    printf("User ID:            %lu\n", info.st_uid);
    printf("Group ID:           %lu\n", info.st_gid);
    printf("File size:          %ld\n", info.st_size);
    printf("Access time:        %s", ctime(&info.st_atime));
    printf("Modification time: %s", ctime(&info.st_mtime));
    printf("Creation time:      %s", ctime(&info.st_ctime));

    return 0;
}
```

This program may print the following:

```
File mode:          0x800
File ID:           0x4574
Volume ref. no.:   0x0
Number of links:    1
User ID:            200
Group ID:           100
File size:          30480
Access time:        Mon Apr 17 19:46:37 1995
Modification time:  Mon Apr 17 19:46:37 1995
Creation time:      Fri Oct  7 12:00:00 1994
```

umask

Sets a UNIX style file creation mask.

```
#include <stat.h> /* Macintosh */  
mode_t umask(mode_t cmask)  
mode_t _umask(mode_t cmask)
```

cmask	mode_t	permission bitmask
-------	--------	--------------------

Remarks

The function `umask` is used for calls to `open()`, `creat()` and `mkdir()` to turn off permission bits in the mode argument.

If `_MSL_POSIX` is true and you are compiling for Macintosh or Windows `umask` returns `mode_t` and takes a `mode_t` otherwise it takes an `int` type.

NOTE The permission bits are not used on either the Mac nor Windows. The function is provided merely to allow compilation and compatibility.

The previous mask. Zero is returned for Mac and Windows operating systems. if `_MSL_POSIX` is on and you are on mac and win `umask` returns `mode_t`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“creat, wcreate” on page 140](#)

[“open, wopen” on page 143](#)

[“mkdir” on page 311](#)

stat.h

Overview of stat.h

stdarg.h

The stdarg.h header file allows the creation of functions that accept a variable number of arguments.

Overview of stdarg.h

The facilities in stdarg.h use for variable arguments are:

- [“va_arg” on page 318](#) a variable argument list
- [“va_copy” on page 318](#)
- [“va_end” on page 319](#) a variable argument end
- [“va_start” on page 320](#) a variable argument start

Variable arguments for functions

The stdarg.h header file allows the creation of functions that accept a variable number of arguments.

A variable-length argument function is defined with an ellipsis (...) as its last argument. For example:

```
int funnyfunc(int a, char c, ...);
```

The function is written using the va_list type, the va_start(), va_arg() and the va_end() macros.

The function has a va_list variable declared within it to hold the list of function arguments. The macro [“va_start” on page 320](#) initializes the va_list variable and is called before gaining access to the arguments. The macro [“va_arg” on page 318](#) returns each of the arguments in va_list. When all the arguments have been processed through va_arg(), the macro [“va_end” on page 319](#) is called to allow a normal return from the function.

va_arg

Macro to return an argument value.

```
#include <stdarg.h>
type va_arg(va_list ap, type type);
```

ap	va_list	A variable list
type	type	The type of the argument value to be obtained

Remarks

The `va_arg()` macro returns the next argument on the function's argument list. The argument returned has the type defined by `type`. The `ap` argument must first be initialized by the `va_start()` macro.

The `va_arg()` macro returns the next argument on the function's argument list of `type`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“va_end” on page 319](#)

[“va_start” on page 320](#)

For example of `va()` usage refer to the example [“Example of va_start\(\) usage.” on page 321](#).

va_copy

Copies and initializes a variable argument list.

```
#include <stdarg.h>
void va_copy(va_list dest, va_list src)
```

dest	va_list	The va_list being initialized
src	va_list	The source va_list being copied

Remarks

The `va_copy()` macro makes a copy of the variable list `src` in a state as if the `va_start` macro had been applied to it followed by the same sequence of `va_arg` macros as had been applied to `src` to bring it into its present state.

There is no return for this facility.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Variable arguments for functions” on page 317](#)

va_end

Prepare a normal function return.

```
#include <stdarg.h>
void va_end(va_list ap);
```

ap	va_list	A variable list
----	---------	-----------------

Remarks

The `va_end()` function cleans the stack to allow a proper function return. The function is called after the function's arguments are accessed with the `va_arg()` macro.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“va_arg” on page 318](#)

[“va_start” on page 320](#)

For example of `va_end` usagr refer to the example [“Example of `va_start\(\)` usage.” on page 321](#).

va_start

Initialize the variable-length argument list.

```
#include <stdarg.h>
void va_start(va_list ap, ParmN Parm);
```

ap	va_list	A variable list
Parm	ParmN	The last named parameter

Remarks

The `va_start()` macro initializes and assigns the argument list to `ap`. The `ParmN` parameter is the last named parameter before the ellipsis (`...`) in the function prototype.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“va_arg” on page 318](#)

[“va_end” on page 319](#)

Listing 30.1 Example of va_start() usage.

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

void multisum(int *dest, ...);

int main(void)
{
    int all;

    all = 0;
    multisum(&all, 13, 1, 18, 3, 0);
    printf("%d\n", all);

    return 0;
}

void multisum(int *dest, ...)
{
    va_list ap;
    int n, sum = 0;

    va_start(ap, dest);

    while ((n = va_arg(ap, int)) != 0)
        sum += n;      /* add next argument to dest */
    *dest = sum;
    va_end(ap);       /* clean things up before leaving */
}
```

Output:

3

stdarg.h

Overview of stdarg.h

stdbool.h

The header `stdbool.h` defines types used for boolean integral values.

Overview of `stdbool.h`

The `stdbool.h` header file consists of defines in [Table 31.1](#) only if the compiler support for c99 has been turned on using the C99 pragma.

```
#pragma c99 on | off | reset
```

Table 31.1 Defines in `stdbool.h`

<code>bool</code>	<code>_Bool</code>
<code>true</code>	1
<code>false</code>	0
<code>__bool_true_false_are_defined</code>	1

Remarks

There are no other defines in this header.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

stdbool.h

Overview of stdbool.h

stddef.h

The stddef.h header file defines commonly used macros and types that are used throughout the ANSI C Standard Library.

Overview of stddef.h

The Commonly used definitions macros and types are defined in stddef.h

- [“NULL” on page 325](#), defines NULL
- [“offsetof” on page 325](#), is the offset of a structure’s member
- [“ptrdiff_t” on page 326](#), used for pointer differences
- [“size_t” on page 326](#), is the return from a sizeof operation
- [“wchar_t” on page 326](#), is a wide character type

NULL

The NULL macro is the null pointer constant used in the Standard Library.

Remarks

This definition may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

offsetof

The offsetof(structure, member) macro expands to an integral expression of type size_t. The value returned is the offset in bytes of a member, from the base of its structure.

NOTE

If the member is a bit field the result is undefined.

Remarks

This macro may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

ptrdiff_t

The `ptrdiff_t` type is the signed integral type used for holding the result of subtracting one pointer's value from another.

Remarks

This type may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

size_t

The `size_t` type is an unsigned integral type returned by the `sizeof()` operator.

Remarks

This type may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

wchar_t

The `wchar_t` type is an integral type capable of holding all character representations of the wide character set.

Remarks

This type may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

stddef.h

Overview of stddef.h

stdint.h

The header `stdint.h` defines types used for standard integral values.

Overview of stdint.h

The `stdint.h` header file consists of integer types listed in the following tables

- [“Integer types” on page 329](#)
- [“Limits of specified-width integer types” on page 331](#)
- [“Macros for integer constants” on page 335](#)
- [“Macros for greatest-width integer constants” on page 335](#)

NOTE

The types in this header may not be implemented on all platforms.
Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Integer types

The header `stdint.h` contains several integer types.

- [“Exact width integer type” on page 330](#)
- [“Minimum width integer type” on page 330](#)
- [“Fastest minimum-width integer types” on page 330](#)
- [“Integer types capable of holding object pointers” on page 330](#)
- [“Greatest width integer types” on page 330](#)
- [“Mac OS X specific integer types” on page 331](#)

Table 33.1 Exact width integer type

Type	Equivalent	Type	Equivalent
int8_t	signed char	int16_t	short int
int32_t	long int	uint8_t	unsigned char
uint16_t	unsigned short int	uint32_t	unsigned long int
int64_t	long long	uint64_t	unsigned long long

Table 33.2 Minimum width integer type

Type	Equivalent	Type	Equivalent
int_least8_t	signed char	int_least16_t	short int
int_least32_t	long int	uint_least8_t	unsigned char
uint_least16_t	unsigned short int	uint_least32_t	unsigned long int
int_least64_t	long long	uint_least64_t	unsigned long long

Table 33.3 Fastest minimum-width integer types

Type	Equivalent	Type	Equivalent
int_fast8_t	signed char	int_fast16_t	short int
int_fast32_t	long int	uint_fast8_t	unsigned char
uint_fast16_t	unsigned short int	uint_fast32_t	unsigned long int
int_fast64_t	long long	uint_fast64_t	unsigned long long

Table 33.4 Integer types capable of holding object pointers

Type	Equivalent	Type	Equivalent
intptr_t	int32_t	uintptr_t	uint32_t

Table 33.5 Greatest width integer types

Type	Equivalent	Type	Equivalent
intmax_t	int64_t	uintmax_t	int32_t

Table 33.6 Mac OS X specific integer types

Type	Equivalent	Type	Equivalent
u_int8_t	unsigned char	u_int16_t	unsigned short
u_int32_t	unsigned int	u_int64_t	unsigned long long
register_t	<code>_std(int32_t)</code>		

Limits of specified-width integer types

The limits of exact-width integer types are defined in the following tables.

- [“Minimum values of exact width signed integer types” on page 331](#)
- [“Maximum values of exact width signed integer types” on page 332](#)
- [“Maximum values of exact width unsigned integer types” on page 332](#)
- [“Minimum values of minimum width signed integer types” on page 332](#)
- [“Maximum values of minimum width signed integer types” on page 332](#)
- [“Maximum values of minimum width unsigned integer types” on page 333](#)
- [“Minimum values of fastest minimum width signed integer types” on page 333](#)
- [“Maximum values of fastest minimum width signed integer types” on page 333](#)
- [“Maximum values of fastest minimum width unsigned integer types” on page 333](#)
- [“Minimum value of pointer holding signed integer types” on page 334](#)
- [“Minimum value of greatest width signed integer type” on page 334](#)
- [“Maximum value of greatest width signed integer type” on page 334](#)
- [“Limits of other integer types” on page 334](#)

Table 33.7 Minimum values of exact width signed integer types

Type	Equivalent
INT8_MIN	SCHAR_MIN
INT16_MIN	SHRT_MIN
INT32_MIN	LONG_MIN
INT64_MIN	LLONG_MIN

Table 33.8 Maximum values of exact width signed integer types

Type	Equivalent
INT8_MAX	SCHAR_MAX
INT16_MAX	SHRT_MAX
INT32_MAX	LONG_MAX
INT64_MAX	LLONG_MAX

Table 33.9 Maximum values of exact width unsigned integer types

Type	Equivalent
UINT8_MAX	UCHAR_MAX
UINT16_MAX	USHRT_MAX
UINT32_MAX	ULONG_MAX
UINT64_MAX	ULLONG_MAX

Table 33.10 Minimum values of minimum width signed integer types

Type	Equivalent
INT_LEAST8_MIN	SCHAR_MIN
INT_LEAST16_MIN	SHRT_MIN
INT_LEAST32_MIN	LONG_MIN
INT_LEAST64_MIN	LLONG_MIN

Table 33.11 Maximum values of minimum width signed integer types

Type	Equivalent
INT_LEAST8_MAX	SCHAR_MAX
INT_LEAST16_MAX	SHRT_MAX
INT_LEAST32_MAX	LONG_MAX
INT_LEAST64_MAX	LLONG_MAX

Table 33.12 Maximum values of minimum width unsigned integer types

Type	Equivalent
UINT_FAST8_MAX	UCHAR_MAX
UINT_FAST16_MAX	USHRT_MAX
UINT_FAST32_MAX	ULONG_MAX
UINT_FAST64_MAX	ULLONG_MAX

Table 33.13 Minimum values of fastest minimum width signed integer types

Type	Equivalent
INT_FAST8_MIN	SCHAR_MIN
INT_FAST16_MIN	SHRT_MIN
INT_FAST32_MIN	LONG_MIN
INT_FAST64_MIN	LLONG_MIN

Table 33.14 Maximum values of fastest minimum width signed integer types

Type	Equivalent
INT_FAST8_MAX	SCHAR_MAX
INT_FAST16_MAX	SHRT_MAX
INT_FAST32_MAX	LONG_MAX
INT_FAST64_MAX	LLONG_MAX

Table 33.15 Maximum values of fastest minimum width unsigned integer types

Type	Equivalent
UINT_FAST8_MAX	UCHAR_MAX
UINT_FAST16_MAX	USHRT_MAX
UINT_FAST32_MAX	ULONG_MAX
UINT_FAST64_MAX	ULLONG_MAX

Table 33.16 Minimum value of pointer holding signed integer types

Type	Equivalent
INTPTR_MIN	LONG_MIN
INTPTR_MAX	LONG_MAX
UINTPTR_MAX	ULONG_MAX

Table 33.17 Minimum value of greatest width signed integer type

Type	Equivalent
INTMAX_MIN	LLONG_MIN

Table 33.18 Maximum value of greatest width signed integer type

Type	Equivalent
UINTMAX_MAX	ULLONG_MAX

Table 33.19 Limits of other integer types

Type	Equivalent
PTRDIFF_MIN	LONG_MIN
PTRDIFF_MAX	LONG_MAX
SIG_ATOMIC_MIN	INT_MIN
SIG_ATOMIC_MAX	INT_MAX

Macros for integer constants

The macros expand to integer constants suitable for initializing objects that have integer types.

```
INT8_C(value)      value
INT16_C(value)     value
INT32_C(value)     value ## L
INT64_C(value)     value ## LL
UINT8_C(value)     value ## U
UINT16_C(value)    value ## U
UINT32_C(value)    value ## UL
UINT64_C(value)   value ## ULL
```

Macros for greatest-width integer constants

The `INTMAX_C` macro expands to an integer constant with the value of its argument and the type `intmax_t`.

```
INTMAX_C(value)    value ## LL
```

The `UINTMAX_C` macro expands to an integer constant with the value of its argument and the type `uintmax_t`.

```
UINTMAX_C(value)   value ## ULL
```

stdint.h

Overview of stdint.h

stdio.h

The `stdio.h` header file provides functions for input/output control.

Overview of stdio.h

The `stdio.h` header file provides functions for input/output control. There are functions for creating, deleting, and renaming files, functions to allow random access, as well as to write and read text and binary data.

The facilities in the `stdio.h` header are:

- [“clearerr” on page 343](#) clears an error from a stream
- [“fclose” on page 346](#) closes a file
- [“fdopen” on page 348](#), converts a file descriptor to a stream
- [“feof” on page 349](#) detects the end of a file
- [“ferror” on page 352](#) checks a file error status
- [“fflush” on page 354](#) flushes a stream
- [“fgetc” on page 356](#) gets a character from a file
- [“fgetpos” on page 358](#) gets a file position from large files
- [“fgets” on page 360](#) gets a string from a file
- [“fileno” on page 361](#) Windows version only
- [“fopen” on page 361](#) opens a file for manipulation
- [“fprintf” on page 365](#) prints formatted output to a file
- [“fputc” on page 371](#) writes a character to a file
- [“fputs” on page 373](#) writes a string to a file
- [“fread” on page 375](#) reads a file
- [“freopen” on page 378](#) reopens a file
- [“fscanf” on page 379](#) scans a file
- [“fseek” on page 387](#) moves to a file position

-
- “[fsetpos](#)” on page 390 sets a file position for large files
 - “[ftell](#)” on page 391 tells a file offset
 - “[fwide](#)” on page 392 determines a character orientation
 - “[fwrite](#)” on page 394 writes to a file
 - “[getc](#)” on page 395 gets a character from a stream
 - “[getchar](#)” on page 397 gets a character from stdin
 - “[gets](#)” on page 398 gets a string from stdin
 - “[perror](#)” on page 400 writes an error to stderr
 - “[printf](#)” on page 402 writes a formatted output to stdout
 - “[putc](#)” on page 410 writes a character to a stream
 - “[putchar](#)” on page 412 writes a character to stdout
 - “[puts](#)” on page 413 writes a string to stdout
 - “[remove](#)” on page 414 removes a file
 - “[rename](#)” on page 416 renames a file
 - “[rewind](#)” on page 417 resets the file indicator to the beginning
 - “[scanf](#)” on page 420 scans stdin for input
 - “[setbuf](#)” on page 425 sets the buffer size for a stream
 - “[setvbuf](#)” on page 428 sets the stream buffer size and scheme
 - “[snprintf](#)” on page 430 writes a number of characters to a buffer
 - “[sprintf](#)” on page 431 to write to a character buffer
 - “[sscanf](#)” on page 432 to scan a string
 - “[tmpfile](#)” on page 434 to create a temporary file
 - “[tmpnam](#)” on page 435 creates temporary name
 - “[ungetc](#)” on page 437 to place a character back in a stream
 - “[vfprintf](#)” on page 440 write variable arguments to file
 - “[vfscanf](#)” on page 442 a variable argument scanf
 - “[vprintf](#)” on page 445 write variable arguments to stdout
 - “[vsnprintf](#)” on page 447 write variable arguments to a char array buffer with a number limit
 - “[vsprintf](#)” on page 449 write variable arguments to a char array buffer
 - “[vsscanf](#)” on page 451 A variable scanf

- “ [wfopen](#)” on page 453 a wide character file open
- “ [wfreopen](#)” on page 453 a wide character file reopen
- “ [wremove](#)” on page 454 a wide character file remove
- “ [wrename](#)” on page 455 a wide character file rename
- “ [wtmpnam](#)” on page 455 a wide character file temporary name

Standard input/output

Streams

A stream is a logical abstraction that isolates input and output operations from the physical characteristics of terminals and structured storage devices. They provide a mapping between a program’s data and the data as actually stored on the external devices. Two forms of mapping are supported, for `text streams` and for `binary streams`. See [“Text Streams and Binary Streams” on page 340](#), for more information.

Streams also provide buffering, which is an abstraction of a file designed to reduce hardware I/O requests. Without buffering, data on an I/O device must be accessed one item at a time. This inefficient I/O processing slows program execution considerably. The `stdio.h` functions use buffers in primary storage to intercept and collect data as it is written to or read from a file. When a buffer is full its contents are actually written to or read from the file, thereby reducing the number of I/O accesses. A buffer’s contents can be sent to the file prematurely by using the `fflush()` function.

The `stdio.h` header offers three buffering schemes: unbuffered, block buffered, and line buffered. The `setvbuf()` function is used to change the buffering scheme of any output stream.

When an output stream is unbuffered, data sent to it are immediately read from or written to the file.

When an output stream is block buffered, data are accumulated in a buffer in primary storage. When full, the buffer’s contents are sent to the destination file, the buffer is cleared, and the process is repeated until the stream is closed. Output streams are block buffered by default if the output refers to a file.

A line buffered output stream operates similarly to a block buffered output stream. Data are collected in the buffer, but are sent to the file when the line is completed with a newline character ('`\n`').

A stream is declared using a pointer to a `FILE`. There are three `FILE` pointers that are automatically opened for a program: `FILE *stdin`, `FILE *stdout`, and `FILE *stderr`. The `FILE` pointers `stdin` and `stdout` are the standard input and output files, respectively, for interactive console I/O. The `stderr` file pointer is the standard error output file, where error messages are written to. The `stderr` stream is written to the console. The `stdin` and `stdout` streams are line buffered while the `stderr` stream is unbuffered.

For more information on routing `stdin`, `stdout`, and `stderr` to a console window, see the chapter on `SIOUX.h` for Macintosh and `WinSIOUX.h` for Windows.

Text Streams and Binary Streams

In a binary stream, there is no transformation of the characters during input or output and what is recorded on the physical device is identical to the program's internal data representation.

A text stream consists of sequence of characters organized into lines, each line terminated by a new-line character. To conform to the host system's convention for representing text on physical devices, characters may have to be added altered or deleted during input and output. Thus, there may not be a one-to-one correspondence between the characters in a stream and those in the external representation. These changes occur automatically as part of the mapping associated with text streams. Of course, the input mapping is the inverse of the output mapping and data that are output and then input through text streams will compare equal with the original data.

In MSL, the text stream mapping affects only the linefeed (LF) character, '`\n`' and the carriage return (CR) character, '`\r`'. The semantics of these two control characters are:

- `\n` Moves the current print location to the start of the next line.
- `\r` Moves the current print location to the start of the current line.

where "current print location" is defined as "that location on a display device where the next character output by the `fputc` function would appear".

The ASCII character set defines the value of LF as `0x0a` and CR as `0x0d` and these are the values that these characters have when they are part of a program's data. On physical devices in the Macintosh operating system, newline characters are represented by `0x0d` and CR as `0x0a`; in other words, the values are interchanged. To meet this requirement, the MSL C library for the Mac, interchanges these values while writing a file and again while reading so that a text stream will be unchanged by

writing to a file and then reading back. MPW chose 0x0a for the newline character in its text file, so, when the MPW switch is on, this interchange of values does not take place. However, if you use this option, you must use the MSL C and C++ libraries that were compiled with this option on.

These versions of the libraries are marked with an N (on 68k) or NL (on PPC), for example ANSI (N/2i) C.68k.Lib or ANSI (NL) C.PPC.Lib. See the notes on the mpwc_newline pragma in the CodeWarrior C Compilers Reference.

On Windows, the situation is different. There, lines are terminated with the character pair CR/LF. As a consequence, in the Windows implementation of MSL, when a text stream is written to a file, a single newline character is converted to the character pair CR/LF and the reverse transformation is made during reading.

The library routines that read a file have no means of determining the mode in which text files were written and thus some assumptions have to be made. On the Mac, it is assumed that the Mac convention is used. Under MPW, it is assumed that the MPW convention is to be used and on Windows, the DOS convention.

File position indicator

The file position indicator is another concept introduced by the `stdio.h` header. Each opened stream has a file position indicator acting as a cursor within a file. The file position indicator marks the character position of the next read or write operation. A read or write operation advances the file position indicator. Other functions are available to adjust the indicator without reading or writing, thus providing random access to a file.

Note that console streams, `stdin`, `stdout`, and `stderr` in particular, do not have file position indicators.

End-of-file and errors

Many functions that read from a stream return the `EOF` value, defined in `stdio.h`. The `EOF` value is a nonzero value indicating that the end-of-file has been reached during the last read or write.

Some `stdio.h` functions also use the `errno` global variable. Refer to the `errno.h` header section. The use of `errno` is described in the relevant function descriptions below.

Wide Character and Byte Character Stream Orientation

There are two types of stream orientation for input and output, a wide character (`wchar_t`) oriented and a byte (`char`) oriented. A stream is without orientation after that stream has been associated with a file, until an operation occurs.

Once any operation is performed on that stream, that stream becomes oriented by that operation to be either byte oriented or wide character oriented and remains that way until the file has been closed and reopened.

After a stream orientation is established, any call to a function of the other orientation is not applied. That is, a byte-oriented input/output function does not have an effect on a wide-oriented stream.

Unicode

Unicode encoded characters are represented and manipulated in MSL as wide characters of type `wchar_t` and can be manipulated with the wide character functions defined in the C Standard.

Table 34.1 The Byte Oriented Functions are:

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

Table 34.2 The wide character Oriented Functions in Wchar.h are

<code>fgetwc</code>	<code>fgetws</code>	<code>fwprintf</code>	<code>fputwc</code>	<code>fputws</code>	<code>fwscanf</code>
<code>getwc</code>	<code>getwchar</code>	<code>putwc</code>	<code>putwchar</code>	<code>swprintf</code>	<code>swscanf</code>
<code>towctrans</code>	<code>vfscanf</code>	<code>vswscanf</code>	<code>vwscanf</code>	<code>vfwprintf</code>	<code>vswprintf</code>
<code>vwprintf</code>	<code>wasctime</code>	<code>watof</code>	<code>wcscat</code>	<code>wcschr</code>	<code>wcsncmp</code>
<code>wcscoll</code>	<code>wcscspn</code>	<code>wcscopy</code>	<code>wcslen</code>	<code>wcsncat</code>	<code>wcsncmp</code>
<code>wcsncpy</code>	<code>wcspbrk</code>	<code>wcsspn</code>	<code>wcsrchr</code>	<code>wcsstr</code>	<code>wcstod</code>

Table 34.2 The wide character Oriented Functions in Wchar.h are

wcstok	wcsftime	wcsxfrm	wctime	wctrans	wmemchr
wmemcmp	wmemcpy	wmemmove	wmemset	wprintf	wscanf

Stream Orientation and Standard Input/Output

The three predefined associated streams, stdin, stdout, and stderr are without orientation at program startup. If any of the standard input/output streams is closed it is not possible to reopen and reconnect that stream to the console. However, it is possible to reopen and connect the stream to a named file.

The C and C++ input/output facilities share the same stdin, stdout and stderr streams.

clearerr

Clear a stream's end-of-file and error status.

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

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The clearerr() function resets the end-of-file status and error status for stream. The end-of-file status and error status are also reset when a stream is opened.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“feof” on page 349](#)
- [“ferror” on page 352](#)
- [“fopen” on page 361](#)

stdio.h*Standard input/output*

[“fseek” on page 387](#)[“rewind” on page 417](#)

Listing 34.1 Example of clearerr() usage

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

int main(void)
{
    FILE *f;

    static char name[] = "myfoo";
    char buf[80];

    // create a file for output
    if ( (f = fopen(name, "w")) == NULL) {
        printf("Can't open %s.\n", name);
        exit(1);
    }
    // output text to the file
    fprintf(f, "chair table chest\n");
    fprintf(f, "desk raccoon\n");

    // close the file
    fclose(f);

    // open the same file again for input
    if ( (f = fopen(name, "r")) == NULL) {
        printf("Can't open %s.\n", name);
        exit(1);
    }

    // read all the text until end-of-file
    for ( ; feof(f) == 0; fgets(buf, 80, f))
        fputs(buf, stdout);

    printf("feof() for file %s is %d.\n", name, feof(f));
    printf("Clearing end-of-file status. . .\n");
    clearerr(f);
    printf("feof() for file %s is %d.\n", name, feof(f));

    // close the file
    fclose(f);

    return 0;
}
```

Output

```
chair table chest
desk raccoon
feof() for file myfoo is 256.
Clearing end-of-file status. . .
feof() for file myfoo is 0.
```

fclose

Close an open file.

```
#include <stdio.h>
int fclose(FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The `fclose()` function closes a file created by `fopen()`, `freopen()`, or `tmpfile()`. The function flushes any buffered data to its file and closes the stream. After calling `fclose()`, `stream` is no longer valid and cannot be used with file functions unless it is reassigned using `fopen()`, `freopen()`, or `tmpfile()`.

All of a program's open streams are flushed and closed when a program terminates normally.

`fclose()` closes then deletes a file created by `tmpfile()`.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fclose()` returns a zero if it is successful and returns an `EOF` if it fails to close a file.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fopen” on page 361](#)

[“freopen” on page 378](#)

[“tmpfile” on page 434](#)

[“abort” on page 459](#)

[“exit” on page 477](#)

Listing 34.2 Example of fclose() usage

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    FILE *f;
    static char name[] = "myfoo";

    // create a new file for output
    if ( (f = fopen(name, "w")) == NULL) {
        printf("Can't open %s.\n", name);
        exit(1);
    }
    // output text to the file
    fprintf(f, "pizza sushi falafel\n");
    fprintf(f, "escargot sprocket\n");

    // close the file
    if (fclose(f) == -1) {
        printf("Can't close %s.\n", name);
        exit(1);
    }

    return 0;
}
```

Output to file myfoo:
pizza sushi falafel
escargot sprocket

fdopen

Converts a file descriptor to a stream.

```
#include <stdio.h>

FILE *fdopen(int fildes, char *mode);
FILE *_fdopen(int fildes, char *mode);
```

fildes	int	A file descriptor, which is integer file number that can be obtained from the function fileno().
mode	char *	The file opening mode

Remarks

This function creates a stream for the file descriptor `fildes`. You can use the stream with such standard I/O functions as `fprintf()` and `getchar()`. In Metrowerks C/C++, it ignores the value of the `mode` argument.

If it is successful, `fdopen()` returns a stream. If it encounters an error, `fdopen()` returns `NULL`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fileno” on page 103](#)

[“open, wopen” on page 143](#)

Listing 34.3 Example of `fdopen()` usage

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

int main(void)
{
    int fd;
    FILE *str;

    fd = open("mytest", O_WRONLY | O_CREAT);

    /* Write to the file descriptor */
    write(fd, "Hello world!\n", 13);
    /* Convert the file descriptor to a stream */

    str = fdopen(fd, "w");

    /* Write to the stream */
    fprintf(str, "My name is %s.\n", getlogin());

    /* Close the stream. */
    fclose(str);
    /* Close the file descriptor */
    close(fd);

    return 0;
}
```

feof

Check the end-of-file status of a stream.

```
#include <stdio.h>

int feof(FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The `feof()` function checks the end-of-file status of the last read operation on stream. The function does not reset the end-of-file status.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`feof()` returns a nonzero value if the stream is at the end-of-file and returns zero if the stream is not at the end-of-file.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“clearerr” on page 343](#)

[“ferror” on page 352](#)

Listing 34.4 Example of feof() usage

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

int main(void)
{
    FILE *f;
    static char filename[80], buf[80] = "";

    // get a filename from the user
    printf("Enter a filename to read.\n");
    gets(filename);

    // open the file for input
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // read text lines from the file until
    // feof() indicates the end-of-file
    for (; feof(f) == 0 ; fgets(buf, 80, f) )
        printf(buf);

    // close the file
    fclose(f);

    return 0;
}
```

Output:

```
Enter a filename to read.
itwerks
The quick brown fox
jumped over the moon.
```

ferror

Check the error status of a stream.

```
#include <stdio.h>
int ferror (FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The `ferror()` function returns the error status of the last read or write operation on `stream`. The function does not reset its error status.

On embedded/RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`ferror()` returns a nonzero value if `stream`'s error status is on, and returns zero if `stream`'s error status is off.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“clearerr” on page 343](#)

[“feof” on page 349](#)

Listing 34.5 Example of ferror() usage

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

int main(void)
{
    FILE *f;
    char filename[80], buf[80];
    int ln = 0;

    // get a filename from the user
    printf("Enter a filename to read.\n");
    gets(filename);

    // open the file for input
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // read the file one line at a time until end-of-file
    do {
        fgets(buf, 80, f);
        printf("Status for line %d: %d.\n", ln++, ferror(f));
    } while (feof(f) == 0);

    // close the file
    fclose(f);

    return 0;
}
```

Output:

```
Enter a filename to read.
itwerks
Status for line 0: 0.
Status for line 1: 0.
Status for line 2: 0.
```

fflush

Empty a stream's buffer to its host environment.

```
#include <stdio.h>
int fflush(FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The `fflush()` function empties `stream`'s buffer to the file associated with `stream`. If the stream points to an output stream or an update stream in which the most recent operation was not input, the `fflush` function causes any unwritten data for that stream to be delivered to the host environment to be written to the file; otherwise the behavior is undefined.

The `fflush()` function should not be used after an input operation.

Using `fflush()` for input streams especially the standard input stream (`stdin`) is undefined and is not supported and will not flush the input buffer.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

The function `fflush()` returns `EOF` if a write error occurs, otherwise it returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“setvbuf” on page 428](#)

Listing 34.6 Example of fflush() usage

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

int main(void)
{
    FILE *f;
    int count;

    // create a new file for output
    if (( f = fopen("foofoo", "w") ) == NULL) {
        printf("Can't open file.\n");
        exit(1);
    }
    for (count = 0; count < 100; count++) {
        fprintf(f, "%5d", count);
        if( (count % 10) == 9 )
        {
            fprintf(f, "\n");
            fflush(f); /* flush buffer every 10 numbers */
        }
    }
    fclose(f);

    return 0;
}
```

Output to file foofoo:

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

fgetc

Read the next character from a stream.

```
#include <stdio.h>
int fgetc(FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The `fgetc()` function reads the next character from `stream` and advances its file position indicator.

On embedded/RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fgetc()` returns the character as an `unsigned char` converted to an `int`. If the end-of-file has been reached or a read error is detected, `fgetc()` returns `EOF`. The difference between a read error and end-of-file can be determined by the use of `feof()` ..

If the file is opened in update mode (+) a file cannot be read from and then written to without repositioning the file using one of the file positioning functions (`fseek()`, `fsetpos()`, or `rewind()`) unless the last read or write reached the end-of-file.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 342](#)
- [“getc” on page 395](#)
- [“getchar” on page 397](#)

Listing 34.7 Example of fgetc() usage

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

int main(void)
{
    FILE *f;
    char filename[80], c;

    // get a filename from the user
    printf("Enter a filename to read.\n");
    gets(filename);

    // open the file for input
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // read the file one character at a time until
    // end-of-file is reached
    while ((c = fgetc(f)) != EOF)
        putchar(c);           // print the character

    // close the file
    fclose(f);

    return 0;
}
```

Output:

Enter a filename to read.

foofoo

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

fgetpos

Get a stream's current file position indicator value.

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);
```

stream	FILE *	A pointer to a FILE stream
pos	fpos_t *	A pointer to a file position type

Remarks

The `fgetpos()` function is used in conjunction with the `fsetpos()` function to allow random access to a file. The `fgetpos()` function gives unreliable results when used with streams associated with a console (`stdin`, `stderr`, `stdout`).

While the `fseek()` and `ftell()` functions use `long` integers to read and set the file position indicator, `fgetpos()` and `fsetpos()` use `fpos_t` values to operate on larger files. The `fpos_t` type, defined in `stdio.h`, can hold file position indicator values that do not fit in a `long int`.

The `fgetpos()` function stores the current value of the file position indicator for `stream` in the `fpos_t` variable `pos` points to.

On embedded/RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fgetpos()` returns zero when successful and returns a nonzero value when it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fseek” on page 387](#)

[“fsetpos” on page 390](#)

[“ftell” on page 391](#)

Listing 34.8 Example of fgetpos() usage

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

int main(void)
{
    FILE *f;
    fpos_t pos;
    char filename[80], buf[80];

    // get a filename from the user
    printf("Enter a filename to read.\n");
    gets(filename);

    // open the file for input
    if (( f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }
    printf("Reading each line twice.\n");

    // get the initial file position indicator value
    // (which is at the beginning of the file)
    fgetpos(f, &pos);

    // read each line until end-of-file is reached
    while (fgets(buf, 80, f) != NULL) {
        printf("Once: %s", buf);

        // move to the beginning of the line to read it again
        fsetpos(f, &pos);
        fgets(buf, 80, f);
        printf("Twice: %s", buf);

        // get the file position of the next line
        fgetpos(f, &pos);
    }

    // close the file
    fclose(f);

    return 0;
}
```

Output:

Enter a filename to read.

myfoo

Reading each line twice.

Once: chair table chest

Twice: chair table chest

Once: desk raccoon

Twice: desk raccoon

fgets

Read a character array from a stream.

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

s	char *	The destination string
n	int	The maximum number of chars read
stream	FILE *	A pointer to a FILE stream

Remarks

The `fgets()` function reads characters sequentially from `stream` beginning at the current file position, and assembles them into `s` as a character array. The function stops reading characters when `n-1` characters have been read. The `fgets()` function finishes reading prematurely if it reaches a newline ('`\n`') character or the end-of-file.

If the file is opened in update mode (+) a file cannot be read from and then written to without repositioning the file using one of the file positioning functions (`fseek()`, `fsetpos()`, or `rewind()`) unless the last read or write reached the end-of-file.

Unlike the `gets()` function, `fgets()` appends the newline character ('`\n`') to `s`. It also null terminates the characters written into the character array.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fgets()` returns a pointer to `s` if it is successful. If it reaches the end-of-file before reading any characters, `s` is untouched and `fgets()` returns a null

pointer (NULL). If an error occurs fgets() returns a null pointer and the contents of s may be corrupted.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“gets” on page 398](#)

For example usage refer to the [“Example of fseek\(\) usage” on page 389](#)

_fileno

This function is described in extras.h as [“_fileno” on page 103](#) in this header it is Windows only.

fopen

Open a file as a stream.

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

filename	const char *	The filename of the file to open
mode	const char *	The file opening mode

Remarks

The fopen() function opens a file specified by filename, and associates a stream with it. The fopen() function returns a pointer to a FILE. This pointer is used to refer to the file when performing I/O operations.

The mode argument specifies how the file is to be used. [“Open modes for fopen\(\),”](#) describes the values for mode.

UPDATE MODE

A file opened with an update mode (“+”) is buffered. The file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the fflush() function or one of the file positioning operations (fseek(), fsetpos(), or rewind()). Similarly, a file cannot be read from and then written to without repositioning the file using one of the file positioning functions unless the last read or write reached the end-of-file.

All file modes, except the append modes (“a”, “a+”, “ab”, “ab+”) set the file position indicator to the beginning of the file. The append modes set the file position indicator to the end-of-file.

NOTE	Write modes, even if in Write and Read (w+, wb+) delete any current data in a file when the file is opened.
-------------	---

Listing 34.9 Open modes for fopen()

Mode	Description
“r”	Open an existing text file for reading only.
“w”	Create a new text file for writing, or open and truncate an existing file
“a”	Open an existing text file, or create a new one if it does not exist, for appending. Writing occurs at the end-of-file position.
“r+”	Update mode. Open an existing text file for reading and writing (See Remarks)
“w+”	Update mode. Create a new text file for writing, or open and truncate an existing file, for writing and reading (See Remarks)
“a+”	Update mode. Open an existing text file or create a new one for reading and writing. Writing occurs at the end-of-file position (See Remarks)
“rb”	Open an existing binary file for reading only.
“wb”	Create a new binary file or open and truncate an existing file, for writing
“ab”	Open an existing binary file, or create a new one if it does not exist, and append. Writing occurs at the end-of-file.

Listing 34.9 Open modes for fopen()

Mode	Description
"r+b" or "rb+"	Update mode. Open an existing binary file for reading and writing (See Remarks)
"w+b" or "wb+"	Update mode. Create a new binary file or open and truncate an existing file, for writing and reading (See Remarks)
"a+b" or "ab+"	Update mode. Open an existing binary file or create a new one for reading and writing. Writing occurs at the end-of-file position (See Remarks)

`fopen()` returns a pointer to a `FILE` if it successfully opens the specified file for the specified operation. `fopen()` returns a null pointer (`NULL`) when it is not successful.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“fclose” on page 346](#)
- [“wfopen” on page 453](#)

Listing 34.10 Example of fopen() usage

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

int main(void)
{
    FILE *f;
    int count;

    // create a new file for output
    if (( f = fopen("foofoo", "w" )) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // output numbers 0 to 9
    for (count = 0; count < 10; count++)
        fprintf(f, "%5d", count);

    // close the file
    fclose(f);

    // open the file to append
    if (( f = fopen("foofoo", "a" )) == NULL) {
        printf("Can't append to file.\n");
        exit(1);
    }

    // output numbers 10 to 19
    for (; count < 20; count++)
        fprintf(f, "%5d\n", count);

    // close file
    fclose(f);

    return 0;
}
```

Output to file foofoo:

0	1	2	3	4	5	6	7	8	9	10
11										
12										
13										
14										
15										

16
17
18
19

fprintf

Send formatted text to a stream.

```
#include <stdio.h>
int fprintf(FILE *stream,
            const char *format, ...);
```

stream	FILE *	A pointer to a FILE stream
format	const char *	The format string

Remarks

The `fprintf()` function writes formatted text to `stream` and advances the file position indicator. Its operation is the same as `printf()` with the addition of the `stream` argument. Refer to the description of `printf()`.

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the `fflush()` function or one of the file positioning operations (`fseek()`, `fsetpos()`, or `rewind()`).

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Output Control String and Conversion Specifiers

The `format` character array contains normal text and conversion specifications. Conversion specifications must have matching arguments in the same order in which they occur in `format`.

The various elements of the format string is specified in the ANSI standards to be in this order from left to right.

- A percent sign

- Optional flags -,+0,# or space
- Optional minimum field width specification
- Optional precision specification
- Optional size specification
- Conversion specifier c,d,e,E,f, Fg,G,i,n,o,p,s,u,x,X or %

A conversion specification describes the format its associated argument is to be converted to. A specification starts with a percent sign (%), optional flag characters, an optional minimum width, an optional precision width, and the necessary, terminating conversion type. Doubling the percent sign (%%) results in the output of a single %.

An optional flag character modifies the formatting of the output; it can be left or right justified, and numerical values can be padded with zeroes or output in alternate forms. More than one optional flag character can be used in a conversion specification. [“Length Modifiers And Conversion Specifiers For Formatted Output Functions” on page 367](#) describes the flag characters.

The optional minimum width is a decimal digit string. If the converted value has more characters than the minimum width, it is expanded as required. If the converted value has fewer characters than the minimum width, it is, by default, right justified (padded on the left). If the - flag character is used, the converted value is left justified (padded on the right).

The maximum minimum field width allowed in MSL Standard Libraries is 509 characters.

The optional precision width is a period character (.) followed by decimal digit string. For floating point values, the precision width specifies the number of digits to print after the decimal point. For integer values, the precision width functions identically to, and cancels, the minimum width specification. When used with a character array, the precision width indicates the maximum width of the output.

A minimum width and a precision width can also be specified with an asterisk (*) instead of a decimal digit string. An asterisk indicates that there is a matching argument, preceding the conversion argument, specifying the minimum width or precision width.

The terminating character, the conversion type, specifies the conversion applied to the conversion specification's matching argument. [“Length Modifiers And Conversion Specifiers For Formatted Output Functions” on page 367](#) describes the conversion type characters.

MSL AltiVec Extensions for Fprintf

The AltiVec extensions to the standard printf family of functions is supported in Metrowerks Standard Libraries.

Separator arguments after % and before any specifier may be any character or may be the @ symbol. The @ symbol is a non-Motorola extension that will use a specified string as a specifier.

In the specific case of a 'c' specifier any char may be used as a separator. For all other specifiers '-' , '+' , '#' , ' ' may not be used.

The listing “[Example of AltiVec Printf Extensions](#)” on page 409 demonstrates their use.

Table 34.3 Length Modifiers And Conversion Specifiers For Formatted Output Functions

Modifier	Description
Size	
h	The h flag followed by d, i, o, u, x, or X conversion specifier indicates that the corresponding argument is a short int or unsigned short int.
l	The lower case L followed by d, i, o, u, x, or X conversion specifier indicates the argument is a long int or unsigned long int. The lower case L followed by a c conversion specifier, indicates that the argument is of type wint_t. The lower case L followed by an s conversion specifier, indicates that the argument is of type wchar_t.
ll	The double l followed by d, i, o, u, x, or X conversion specifier indicates the argument is a long long or unsigned long long
L	The upper case L followed by e, E, f, g, or G conversion specifier indicates a long double.
v	AltiVec: A vector bool char, vector signed char or vector unsigned char when followed by c, d, i, o, u, x or X A vector float, when followed by f.
vh hv	AltiVec: A vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X

Table 34.3 Length Modifiers And Conversion Specifiers For Formatted Output Functions

vl	AltiVec: A vector int, vector unsigned int or vector bool int when followed by c, d, i, o, u, x or X
----	---

Flags

-	The conversion will be left justified.
+	The conversion, if numeric, will be prefixed with a sign (+ or -). By default, only negative numeric values are prefixed with a minus sign (-).
space	If the first character of the conversion is not a sign character, it is prefixed with a space. Because the plus sign flag character (+) always prefixes a numeric value with a sign, the space flag has no effect when combined with the plus flag.
#	For c, d, i, and u conversion types, the # flag has no effect. For s conversion types, a pointer to a Pascal string, is output as a character string. For o conversion types, the # flag prefixes the conversion with a 0. For x conversion types with this flag, the conversion is prefixed with a 0x. For e, E, f, g, and G conversions, the # flag forces a decimal point in the output. For g and G conversions with this flag, trailing zeroes after the decimal point are not removed.
0	This flag pads zeroes on the left of the conversion. It applies to d, i, o, u, x, X, e, E, f, g, and G conversion types. The leading zeroes follow sign and base indication characters, replacing what would normally be space characters. The minus sign flag character overrides the 0 flag character. The 0 flag is ignored when used with a precision width for d, i , o, u, x, and X conversion types.
@	AltiVec: This flag indicates a pointer to a string specified by an argument. This string will be used as a separator for vector elements.

Conversions

d	The corresponding argument is converted to a signed decimal.
i	The corresponding argument is converted to a signed decimal.
o	The argument is converted to an unsigned octal.
u	The argument is converted to an unsigned decimal.

Table 34.3 Length Modifiers And Conversion Specifiers For Formatted Output Functions

x, X	The argument is converted to an unsigned hexadecimal. The x conversion type uses lowercase letters (abcdef) while X uses uppercase letters (ABCDEF).
n	This conversion type stores the number of items output by printf() so far. Its corresponding argument must be a pointer to an int.
f, F	The corresponding floating point argument (float, or double) is printed in decimal notation. The default precision is 6 (6 digits after the decimal point). If the precision width is explicitly 0, the decimal point is not printed. For the f conversion specifier, a double argument representing infinity produces [-]inf; a double argument representing a NaN (Not a number) produces [-]nan. For the F conversion specifier, [-]INF or [-]NAN are produced instead.
e, E	The floating point argument (float or double) is output in scientific notation: [-]b.aaaee. There is one digit (b) before the decimal point. Unless indicated by an optional precision width, the default is 6 digits after the decimal point (aaa). If the precision width is 0, no decimal point is output. The exponent (ee) is at least 2 digits long. The e conversion type uses lowercase e as the exponent prefix. The E conversion type uses uppercase E as the exponent prefix.
g, G	The g conversion type uses the f or e conversion types and the G conversion type uses the F or E conversion types. Conversion type e (or E) is used only if the converted exponent is less than -4 or greater than the precision width. The precision width indicates the number of significant digits. No decimal point is output if there are no digits following it.
c	The corresponding argument is output as a character.
s	The corresponding argument, a pointer to a character array, is output as a character string. Character string output is completed when a null character is reached. The null character is not output.
p	The corresponding argument is taken to be a pointer. The argument is output using the x conversion type format.

Table 34.3 Length Modifiers And Conversion Specifiers For Formatted Output Functions

	CodeWarrior Extensions
#s	The corresponding argument, a pointer to a Pascal string, is output as a character string. A Pascal character string is a length byte followed by the number characters specified in the length byte. Note: This conversion type is an extension to the ANSI C library but applied in the same manner as for other format variations.

`fprintf()` returns the number of arguments written or a negative number if an error occurs.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“printf” on page 402](#)

[“sprintf” on page 431](#)

[“vfprintf” on page 440](#)

[“vprintf” on page 445](#)

[“vsprintf” on page 449](#)

Listing 34.11 Example of `fprintf()` usage

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

int main(void)
{
    FILE *f;
    static char filename[] = "myfoo";
    int a = 56;
    char c = 'M';
    double x = 483.582;

    // create a new file for output
    if ((f = fopen(filename, "w")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // output formatted text to the file
    fprintf(f, "%10s %4.4f %-10d\n%10c", filename, x, a, c);

    // close the file
    fclose(f);

    return 0;
}
```

Output to file foo:
myfoo 483.5820 56
M

fputc

Write a character to a stream.

```
#include <stdio.h>

int fputc(int c, FILE *stream);
```

c	int	The character to write to a file
stream	FILE *	A pointer to a FILE stream

Remarks

The `fputc()` function writes the character `c` to `stream` and advances `stream`'s file position indicator. Although the `c` argument is an `unsigned int`, it is converted to a `char` before being written to `stream`. `fputc()` is written as a function, not as a macro.

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the `fflush()` function or one of the file positioning operations (`fseek()`, `fsetpos()`, or `rewind()`).

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fputc()` returns the character written if it is successful, and returns `EOF` if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“putc” on page 410](#)

[“putchar” on page 412](#)

Listing 34.12 Example of fputc() usage

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

int main(void)
{
    FILE *f;
    int letter;

    // create a new file for output
    if (( f = fopen("foofoo", "w") ) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // output the alphabet to the file one letter
    // at a time
    for (letter = 'A'; letter <= 'Z'; letter++)
        fputc(letter, f);
    fclose(f);

    return 0;
}
```

Output to file foofoo:
ABCDEFIGHIJKLMNOPQRSTUVWXYZ

fputs

Write a character array to a stream.

```
#include <stdio.h>

int fputs(const char *s, FILE *stream);
```

s	const char *	The string to write to a file
stream	FILE *	A pointer to a FILE stream

Remarks

The `fputs()` function writes the array pointed to by `s` to `stream` and advances the file position indicator. The function writes all characters in `s` up to, but not including, the terminating null character. Unlike `puts()`, `fputs()` does not terminate the output of `s` with a newline ('`\n`').

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the `fflush()` function or one of the file positioning operations (`fseek()`, `fsetpos()`, or `rewind()`).

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fputs()` returns a zero if successful, and returns a nonzero value when it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“puts” on page 413](#)

Listing 34.13 Example of fputs() usage

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

int main(void)
{
    FILE *f;

    // create a new file for output
    if (( f = fopen("foofoo", "w") ) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // output character strings to the file
    fputs("undo\n", f);
    fputs("copy\n", f);
    fputs("cut\n", f);
    fputs("rickshaw\n", f);

    // close the file
    fclose(f);

    return 0;
}
```

```
Output to file foofoo:
undo
copy
cut
rickshaw
```

fread

Read binary data from a stream.

```
#include <stdio.h>

size_t fread(void *ptr, size_t size,
            size_t nmemb, FILE *stream);
```

ptr	void *	A pointer to the read destination
size	size_t	The size of the array elements pointed to
nmemb	size_t	Number of elements to be read
stream	FILE *	A pointer to a FILE stream

Remarks

The `fread()` function reads a block of binary or text data and updates the file position indicator. The data read from `stream` are stored in the array pointed to by `ptr`. The `size` and `nmemb` arguments describe the size of each item and the number of items to read, respectively.

The `fread()` function reads `nmemb` items unless it reaches the end-of-file or a read error occurs.

If the file is opened in update mode (+) a file cannot be read from and then written to without repositioning the file using one of the file positioning functions (`fseek()`, `fsetpos()`, or `rewind()`) unless the last read or write reached the end-of-file.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fread()` returns the number of items read successfully.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fgets” on page 360](#)

[“fwrite” on page 394](#)

Listing 34.14 Example of fread() usage

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

// define the item size in bytes
#define BUFSIZE 40

int main(void)
{
    FILE *f;
    static char s[BUFSIZE] = "The quick brown fox";
    char target[BUFSIZE];

    // create a new file for output and input
    if (( f = fopen("foo", "w+")) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // output to the stream using fwrite()
    fwrite(s, sizeof(char), BUFSIZE, f);

    // move to the beginning of the file
    rewind(f);

    // now read from the stream using fread()
    fread(target, sizeof(char), BUFSIZE, f);

    // output the results to the console
    puts(s);
    puts(target);

    // close the file
    fclose(f);

    return 0;
}
```

Output:

```
The quick brown fox
The quick brown fox
```

freopen

Re-direct a stream to another file.

```
#include <stdio.h>

FILE *freopen(const char *filename,
const char *mode, FILE *stream);
```

filename	const char *	The name of the file to re-open
mode	const char *	The file opening mode
stream	FILE *	A pointer to a FILE stream

Remarks

The `freopen()` function changes the file that `stream` is associated with to another file. The function first closes the file the stream is associated with, and opens the new file, `filename`, with the specified `mode`, using the same stream.

`fopen()` returns the value of `stream`, if it is successful. If `fopen()` fails it returns a null pointer (`NULL`).

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“fopen” on page 361](#)

Listing 34.15 Example of freopen() usage

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

int main(void)
{
    FILE *f;

    // re-direct output from the console to a new file
    if (( f = freopen("newstdout", "w+", stdout)) == NULL) {
        printf("Can't create new stdout file.\n");
        exit(1);
    }
    printf("If all goes well, this text should be in\n");
    printf("a text file, not on the screen via stdout.\n");
    fclose(f);

    return 0;
}
```

fscanf

Read formatted text from a stream.

```
#include <stdio.h>

int fscanf(FILE *stream, const char *format, ...);
```

stream	FILE *	A pointer to a FILE stream
format	const char *	A format string

Remarks

The `fscanf()` function reads programmer-defined, formatted text from `stream`. The function operates identically to the `scanf()` function with the addition of the `stream` argument indicating the stream to read from. Refer to the `scanf()` function description.

If the file is opened in update mode (+) a file cannot be read from and then written to without repositioning the file using one of the file positioning

functions (fseek(), fsetpos(), or rewind()) unless the last read or write reached the end-of-file.

On embedded/RTOS systems this function only is implemented for stdin, stdout and stderr files.

Input Control String and Conversion Specifiers

The `format` argument is a character array containing normal text, white space (space, tab, newline), and conversion specifications. The normal text specifies literal characters that must be matched in the input stream. A white space character indicates that white space characters are skipped until a non-white space character is reached. The conversion specifications indicate what characters in the input stream are to be converted and stored.

The conversion specifications must have matching arguments in the order they appear in `format`. Because `scanf()` stores data in memory, the arguments matching the conversion specification arguments must be pointers to objects of the relevant types.

A conversion specification consists of the percent sign (%) prefix, followed by an optional maximum width or assignment suppression, and ending with a conversion type. A percent sign can be skipped by doubling it in `format`; %% signifies a single % in the input stream.

An optional width is a decimal number specifying the maximum width of an input field. `scanf()` will not read more characters for a conversion than is specified by the width.

An optional assignment suppression character (*) can be used to skip an item by reading it but not assigning it. A conversion specification with assignment suppression must not have a corresponding argument.

The last character, the conversion type, specifies the kind of conversion requested. [“Length Modifiers And Conversion Specifiers For Formatted Input” on page 382](#), describes the conversion type characters.

MSL AltiVec Extensions for Scanf

The AltiVec extensions to the standard `scanf` family of functions is supported in Metrowerks Standard Libraries.

Separator arguments after % and before any specifier may be any character or may be the @ symbol. The @ symbol is a non-Motorola extension that will use a specified string as a specifier.

In the specific case of a 'c' specifier any char may be used as a separator. For all other specifiers ',', '+', '#', '' may not be used.

The listing [“Example of AltiVec Scanf Extensions” on page 425](#) demonstrates their use.

Table 34.4 Length Modifiers And Conversion Specifiers For Formatted Input

Modifier	Description
Length Specifiers	
hh	The hh flag indicates that the following d, i, o, u, x, X or n conversion specifier applies to an argument that is of type char or unsigned char.
h	The h flag indicates that the following d, i, o, u, x, X or n conversion specifier applies to an argument that is of type short int or unsigned short int.
I	When used with integer conversion specifier, the I flag indicates long int or an unsigned long int type. When used with floating point conversion specifier, the I flag indicates a double. When used with a c or s conversion specifier, the I flag indicates that the corresponding argument with type pointer to wchar_t.
ll	When used with integer conversion specifier, the ll flag indicates that the corresponding argument is of type long long or an unsigned long long.
L	The L flag indicates that the corresponding float conversion specifier corresponds to an argument of type long double.
v	AltiVec: A vector bool char, vector signed char or vector unsigned char when followed by c, d, i, o, u, x or X A vector float, when followed by f.
vh hv	AltiVec: vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X
vl lv	AltiVec: vector long, vector unsigned long or vector bool when followed by c, d, i, o, u, x or X
Conversion Specifiers	
d	A decimal integer is read.
i	A decimal, octal, or hexadecimal integer is read. The integer can be prefixed with a plus or minus sign (+, -), 0 for octal numbers, 0x or 0X for hexadecimal numbers.
o	An octal integer is read.
u	An unsigned decimal integer is read.

x, X	A hexadecimal integer is read.
e, E, f, g, G	A floating point number is read. The number can be in plain decimal format (e.g. 3456 . 483) or in scientific notation ([-] b.aaa [-] dd) .
s	A character string is read. The input character string is considered terminated when a white space character is reached or the maximum width has been reached. The null character is appended to the end of the array.
c	A character is read. White space characters are not skipped, but read using this conversion specifier.
p	A pointer address is read. The input format should be the same as that output by the p conversion type in printf().
n	This conversion type does not read from the input stream but stores the number of characters read so far in its corresponding argument.
[scanf]	Input stream characters are read and filtered determined by the scanf. See " Scanset " on page 383, for a full description.

Scanset

The conversion specifier %[allows you to specify a scanset, which is a sequence of characters that will be read and stored in the string pointed to by the scanset's corresponding argument. The characters between the [and the terminating] define the scanset. A null character is appended to the end of the character sequence.

Input stream characters are read until a character is found that is not in the scanset. If the first character of scanset is a circumflex (^) then input stream characters are read until a character from the scanset is read. A null character is appended to the end of the character array.

Thus, the conversion specifier %[abcdef] specifies that the scanset is abcdef and any of the characters 'a' through 'f' are to be accepted and stored. As soon as any character outside this set is encountered, reading and storing will cease. Thus, for example, assuming we have the declaration:

```
char str[20];  
  
the execution of  
  
sscanf("acdfxbe", "%[abcdef]", str);
```

will store acdf in str; the 'x' and following characters will not be stored because the 'x' is not in the scanset.

If the first character of the scanset is the circumflex, ^, then the following characters will define a set of characters such that encountering any one of them will cause reading and storing to stop; any character outside a scanset defined in this way will be accepted, we will call this an exclusionary scanset. Thus execution of

```
sscanf("stuvawxyz", "%[^abcdef]", str);
```

will store stuv in str. If you want ^ to be part of the scanset, you cannot list it as the first character otherwise it will be interpreted as introducing the members of an exclusionary scanset. Thus %[^abc] defines the exclusionary scanset abc whereas %[a^bc] defines the scanset abc^. %[^a^bc] defines the exclusionary scanset abc^, as does %[^^abc].

If you want] to be in the scanset, it must be the first character of the scanset, immediately following the %[or, to be in an exclusionary scanset, immediately after the ^, for example, %[]abc] or %[^]abc]. In any other position, the] will be interpreted as terminating the scanset.

To include the - character in the scanset, it must be either listed first (possibly after an initial ^ or last, thus for example, %[-abc], %[abc-], %[^-abc], or %[^abc-]). The C Standard explicitly states:

- If a - character is in the scanlist and is not the first, nor the second where the first character is a ^, nor the last character, the behavior is implementation-defined.

MSL interprets such a use of - in a scanlist as defining a range of characters; thus, the specification %[a-z] as being the equivalent of %[abcdefghijklmnopqrstuvwxyz]. You should bear in mind that this is MSL's interpretation and such usage may be interpreted differently in other C library implementations. Note also that it is assumed that the numeric value of the character before the - is less than that of the one after. If this relationship does not hold undefined and probably unwanted effects may be experienced.

fscanf() returns the number of items items read or, if an input error occurs before any conversions, the value EOF. If there is an error in reading data that is inconsistent with the format string, fscanf() sets errno to a nonzero value. fscanf() returns EOF if it reaches the end-of-file.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“errno” on page 95](#)

[“scanf” on page 420](#)

Listing 34.16 Example of fscanf() usage

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

int main(void)
{
    FILE *f;
    int i;
    double x;
    char c;

    // create a new file for output and input
    if (( f = fopen("foobar", "w+")) == NULL) {
        printf("Can't create new file.\n");
        exit(1);
    }

    // output formatted text to the file
    fprintf(f, "%d\n%lf\n%c\n", 45, 983.3923, 'M');

    // go to the beginning of the file
    rewind(f);

    // read from the stream using fscanf()
    fscanf(f, "%d %lf %c", &i, &x, &c);

    // close the file
    fclose(f);

    printf("The integer read is %d.\n", i);
    printf("The floating point value is %f.\n", x);
    printf("The character is %c.\n", c);

    return 0;
}
```

Output:

```
The integer read is 45.
The floating point value is 983.392300.
The character is M.
```

fseek

Move the file position indicator.

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

stream	FILE *	A pointer to a FILE stream
offset	long	The offset to move in bytes
whence	int	The starting position of the offset

Remarks

The `fseek()` function moves the file position indicator to allow random access to a file.

The function moves the file position indicator either absolutely or relatively. The `whence` argument can be one of three values defined in `stdio.h`: `SEEK_SET`, `SEEK_CUR`, `SEEK_END`.

The `SEEK_SET` value causes the file position indicator to be set `offset` bytes from the beginning of the file. In this case `offset` must be equal or greater than zero.

The `SEEK_CUR` value causes the file position indicator to be set `offset` bytes from its current position. The `offset` argument can be a negative or positive value.

The `SEEK_END` value causes the file position indicator to be set `offset` bytes from the end of the file. The `offset` argument must be equal or less than zero.

The `fseek()` function undoes the last `ungetc()` call and clears the end-of-file status of `stream`.

NOTE

The function `fseek` has limited use when used with MS DOS text files opened in `text` mode because of the carriage return / line feed translations. For more information review “[Text Streams and Binary Streams” on page 340](#).

The `fseek` operations may be incorrect near the end of the file due to eof translations.

The only `fseek` operations guaranteed to work in MS DOS text files opened in text mode are:

- Using the offset returned from `ftell()` and seeking from the beginning of the file.
- Seeking with an offset of zero from `SEEK_SET`, `SEEK_CUR` and `SEEK_END`.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fseek()` returns zero if it is successful and returns a nonzero value if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fgetpos” on page 358](#)

[“fsetpos” on page 390](#)

[“ftell” on page 391](#)

Listing 34.17 Example of fseek() usage

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

int main(void)
{
    FILE *f;
    long int pos1, pos2, newpos;
    char filename[80], buf[80];

    // get a filename from the user
    printf("Enter a filename to read.\n");
    gets(filename);

    // open a file for input
    if (( f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    printf("Reading last half of first line.\n");

    // get the file position indicator before and after
    // reading the first line
    pos1 = ftell(f);
    fgets(buf, 80, f);
    pos2 = ftell(f);
    printf("Whole line: %s\n", buf);

    // calculate the middle of the line
    newpos = (pos2 - pos1) / 2;

    fseek(f, newpos, SEEK_SET);
    fgets(buf, 80, f);
    printf("Last half: %s\n", buf);

    // close the file
    fclose(f);

    return 0;
}
```

Output:
Enter a filename to read.
itwerks

Reading last half of first line.
Whole line: The quick brown fox

Last half: brown fox

fsetpos

Set the file position indicator.

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);
```

stream	FILE *	A pointer to a FILE stream
pos	fpos_t	A pointer to a file positioning type

Remarks

The `fsetpos()` function sets the file position indicator for `stream` using the value pointed to by `pos`. The function is used in conjunction with `fgetpos()` when dealing with files having sizes greater than what can be represented by the `long int` argument used by `fseek()`.

`fsetpos()` undoes the previous call to `ungetc()` and clears the end-of-file status.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fsetpos()` returns zero if it is successful and returns a nonzero value if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fgetpos” on page 358](#)

[“fseek” on page 387](#)

[“ftell” on page 391](#)

For example usage refer to [“Example of fgetpos\(\) usage” on page 359](#)

ftell

Return the current file position indicator value.

```
#include <stdio.h>
long int ftell(FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The `ftell()` function returns the current value of stream's file position indicator. It is used in conjunction with `fseek()` to provide random access to a file.

The function will not work correctly when it is given a stream associated to a console file, such as `stdin`, `stdout`, or `stderr`, where a file indicator position is not applicable. Also, `ftell()` cannot handle files with sizes larger than what can be represented with a `long int`. In such a case, use the `fgetpos()` and `fsetpos()` functions.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`ftell()`, when successful, returns the current file position indicator value. If it fails, `ftell()` returns `-1L` and sets the global variable `errno` to a nonzero value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“errno” on page 95](#), [“fgetpos” on page 358](#)

For example usage refer to [“Example of fseek\(\) usage” on page 389](#)

fwide

Determine the orientation of a stream.

```
#include <stdio.h>
int fwide(FILE *stream, int orientation);
```

stream	FILE *	A pointer to the stream being tested
orientation	int	The desired orientation

Remarks

The `fwide` function determines the orientation of the stream pointed to by `stream`. If the value of `orientation` is greater than zero and `stream` is without orientation, `stream` is made to be wide oriented. If the value of `orientation` is less than zero and `stream` is without orientation, `stream` is made to be byte oriented. Otherwise, the value of `orientation` is zero and the function does not alter the orientation of the stream. In all cases, if `stream` already has an orientation, it will not be changed.

The `fwide` function returns a value greater than zero if, after the call, the stream has wide orientation, a value less than zero if the stream has byte orientation, or zero if the stream has no orientation.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 34.18 Example of fwide() usage.

```
#include <stdio.h>

int main()
{
    FILE * fp;
    char filename[FILENAME_MAX];
    int orientation;
    char * cptr;

    cptr = tmpnam(filename);
    fp = fopen(filename, "w");
    orientation = fwide(fp, 0);

    // A newly opened file has no orientation
    printf("Initial orientation = %i\n", orientation);
    fprintf(fp, "abcdefghijklmnopqrstuvwxyz\n");

    // A byte oriented output operation will set the orientation
    // to byte oriented
    orientation = fwide(fp, 0);

    printf("Orientation after fprintf = %i\n", orientation);
    fclose(fp);
    fp = fopen(filename, "r");
    orientation = fwide(fp, 0);
    printf("Orientation after reopening = %i\n", orientation);
    orientation = fwide(fp, -1);

    // fwide with a non-zero orientation argument will set an
    // unoriented file's orientation
    printf("Orientation after fwide = %i\n", orientation);
    orientation = fwide(fp, 1);

    // but will not change the file's orientation if it
    // already has an orientation
    printf("Orientation after second fwide = %i\n", orientation);
    fclose(fp);
    remove(filename);

    return 0;
}
```

Output:

Initial orientation = 0

```
Orientation after fprintf = -1
Orientation after reopening = 0
Orientation after fwide   = -1
Orientation after second fwide = -1
```

fwrite

Write binary data to a stream.

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size,
             size_t nmemb, FILE *stream);
```

ptr	void *	A pointer to the item being written
size	size_t	The size of the item being written
nmemb	size_t	The number of items being written
stream	FILE *	A pointer to a FILE stream

Remarks

The `fwrite()` function writes `nmemb` items of `size` bytes each to `stream`. The items are contained in the array pointed to by `ptr`. After writing the array to `stream`, `fwrite()` advances the file position indicator accordingly.

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the `fflush()` function or one of the file positioning operations (`fseek()`, `fsetpos()`, or `rewind()`).

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`fwrite()` returns the number of items successfully written to `stream`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fread” on page 375](#)

For example of fwrite() usage refer to [“Example of fread\(\) usage” on page 377](#)

getc

Read the next character from a stream.

```
#include <stdio.h>
int getc(FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The `getc()` function reads the next character from `stream`, advances the file position indicator, and returns the character as an `int` value. Unlike the `fgetc()` function, `getc()` is implemented as a macro.

If the file is opened in update mode (+) it cannot be read from and then written to without being repositioned using one of the file positioning functions (`fseek()`, `fsetpos()`, or `rewind()`) unless the last read or write reached the end-of-file.

`getc()` returns the next character from the stream or returns `EOF` if the end-of-file has been reached or a read error has occurred.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 342](#)
- [“fgetc” on page 356](#)
- [“fputc” on page 371](#)
- [“getchar” on page 397](#)
- [“putchar” on page 412](#)

stdio.h*Standard input/output*

Listing 34.19 Example of getc() usage.

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

int main(void)
{
    FILE *f;
    char filename[80], c;

    // get a filename from the user
    printf("Enter a filename to read.\n");
    scanf("%s", filename);

    // open a file for input
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // read one character at a time until end-of-file
    while ((c = getc(f)) != EOF)
        putchar(c);

    // close the file
    fclose(f);

    return 0;
}
```

Output

```
Enter a filename to read.
foofoo
   0   1   2   3   4   5   6   7   8   9
 10   11  12  13  14  15  16  17  18  19
 20   21  22  23  24  25  26  27  28  29
 30   31  32  33  34  35  36  37  38  39
 40   41  42  43  44  45  46  47  48  49
 50   51  52  53  54  55  56  57  58  59
 60   61  62  63  64  65  66  67  68  69
 70   71  72  73  74  75  76  77  78  79
 80   81  82  83  84  85  86  87  88  89
 90   91  92  93  94  95  96  97  98  99
```

getchar

Get the next character from `stdin`.

```
#include <stdio.h>
int getchar(void);
```

Remarks

The `getchar()` function reads a character from the `stdin` stream.

The function `getchar()` is implemented as `getc(stdin)` and as such `getchar`'s return may be delayed or optimized out of program order if `stdin` is buffered. For most implementations `stdin` is line buffered.

`getchar()` returns the value of the next character from `stdin` as an `int` if it is successful. `getchar()` returns `EOF` if it reaches an end-of-file or an error occurs.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See also:

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fgetc” on page 356](#)

[“getc” on page 395](#)

[“putchar” on page 412](#)

stdio.h*Standard input/output*

Listing 34.20 Example of getchar() usage

```
#include <stdio.h>

int main(void)
{
    int c;

    printf("Enter characters to echo, * to quit.\n");

    // characters entered from the console are echoed
    // to it until a * character is read
    while ( (c = getchar()) != '*' )
        putchar(c);

    printf("\nDone!\n");

    return 0;
}
```

Output:

```
Enter characters to echo, * to quit.
I'm experiencing deja-vu *
I'm experiencing deja-vu
Done!
```

gets

Read a character array from stdin.

```
#include <stdio.h>

char *gets(char *s);
```

s	char s	The string being written in to
---	--------	--------------------------------

Remarks

The `gets()` function reads characters from `stdin` and stores them sequentially in the character array pointed to by `s`. Characters are read until either a newline or an end-of-file is reached.

Unlike `fgets()`, the programmer cannot specify a limit on the number of characters to read. Also, `gets()` reads and ignores the newline character ('`\n`') so that it can advance the file position indicator to the next line. The newline character is not stored in `s`. Like `fgets()`, `gets()` terminates the character string with a null character.

If an end-of-file is reached before any characters are read, `gets()` returns a null pointer (`NULL`) without affecting the character array at `s`. If a read error occurs, the contents of `s` may be corrupted.

`gets()` returns `s` if it is successful and returns a null pointer if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fgets” on page 360](#)

stdio.h*Standard input/output*

Listing 34.21 Example of gets() usage.

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

int main(void)
{
    char buf[100];

    printf("Enter text lines to echo.\n");
    printf("Enter an empty line to quit.\n");

    // read character strings from the console
    // until an empty line is read
    while (strlen(gets(buf)) > 0)
        puts(buf);    // puts() appends a newline to its output

    printf("Done!\n");

    return 0;
}
```

Output:

```
Enter text lines to echo.
Enter an empty line to quit.
I'm experiencing deja-vu
I'm experiencing deja-vu
Now go to work
Now go to work
```

```
Done!
```

perror

Output an error message to stderr.

```
#include <stdio.h>

void perror(const char *s);
```

s	const char *	Prints an errno and message
---	--------------	-----------------------------

Remarks

If *s* is not NULL or a pointer to a null string the *perror()* function outputs to *stderr* the character array pointed to by *s* followed by a colon and a space ': '. Then, the error message that would be returned by *strerror()* for the current value of the global variable *errno*.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“abort” on page 459](#)

[“errno” on page 95](#)

Listing 34.22 Example of perror() usage.

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

int main()
{
    perror("No error reported as");
    errno = EDOM;
    perror("Domain error reported as");
    errno = ERANGE;
    perror("Range error reported as");

    return 0;
}
```

Output

```
No error reported as: No Error
Domain error reported as: Domain Error
Range error reported as: Range Error
```

printf

Output formatted text.

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

format	const char *	A format string
--------	--------------	-----------------

Remarks

The `printf()` function outputs formatted text. The function takes one or more arguments, the first being `format`, a character array pointer. The optional arguments following `format` are items (integers, characters, floating point values, etc.) that are to be converted to character strings and inserted into the output of `format` at specified points.

The `printf()` function sends its output to `stdout`.

Printf Control String and Conversion Specifiers

The `format` character array contains normal text and conversion specifications. Conversion specifications must have matching arguments in the same order in which they occur in `format`.

The various elements of the format string is specified in the ANSI standards to be in this order from left to right.

- A percent sign
- Optional flags `-,+0,#` or space
- Optional minimum field width specification
- Optional precision specification
- Optional size specification
- Conversion specifier `c,d,e,E,f,F,g,G,i,n,o,p,s,u,x,X` or `%`

A conversion specification describes the format its associated argument is to be converted to. A specification starts with a percent sign (%), optional flag characters, an optional minimum width, an optional precision width, and the necessary, terminating conversion type. Doubling the percent sign (%%) results in the output of a single %.

An optional flag character modifies the formatting of the output; it can be left or right justified, and numerical values can be padded with zeroes or output in alternate forms. More than one optional flag character can be used in a conversion specification. [“Length Modifiers And Conversion Specifiers For Formatted Output Functions” on page 404](#) describes the flag characters.

The optional minimum width is a decimal digit string. If the converted value has more characters than the minimum width, it is expanded as required. If the converted value has fewer characters than the minimum width, it is, by default, right justified (padded on the left). If the - flag character is used, the converted value is left justified (padded on the right).

The maximum minimum field width allowed in MSL Standard Libraries is 509 characters.

The optional precision width is a period character (.) followed by decimal digit string. For floating point values, the precision width specifies the number of digits to print after the decimal point. For integer values, the precision width functions identically to, and cancels, the minimum width specification. When used with a character array, the precision width indicates the maximum width of the output.

A minimum width and a precision width can also be specified with an asterisk (*) instead of a decimal digit string. An asterisk indicates that there is a matching argument, preceding the conversion argument, specifying the minimum width or precision width.

The terminating character, the conversion type, specifies the conversion applied to the conversion specification's matching argument. [“Length Modifiers And Conversion Specifiers For Formatted Output Functions” on page 404](#), describes the conversion type characters.

MSL AltiVec Extensions for Printf

The AltiVec extensions to the standard printf family of functions is supported in Metrowerks Standard Libraries.

Separator arguments after % and before any specifier may be any character or may be the @ symbol. The @ symbol is a non-Motorola extension that will use a specified string as a specifier.

In the specific case of a 'c' specifier any char may be used as a separator. For all other specifiers '-' , '+' , '#' , '' may not be used.

The listing [“Example of AltiVec Printf Extensions” on page 409](#) demonstrates their use.

Table 34.5 Length Modifiers And Conversion Specifiers For Formatted Output Functions

Modifier	Description
Size	
h	The h flag followed by d, i, o, u, x, or X conversion specifier indicates that the corresponding argument is a short int or unsigned short int.
l	The lower case L followed by d, i, o, u, x, or X conversion specifier indicates the argument is a long int or unsigned long int. The lower case L followed by a c conversion specifier, indicates that the argument is of type wint_t. The lower case L followed by an s conversion specifier, indicates that the argument is of type wchar_t.
ll	The double l followed by d, i, o, u, x, or X conversion specifier indicates the argument is a long long or unsigned long long
L	The upper case L followed by e, E, f, g, or G conversion specifier indicates a long double.
v	AltiVec: A vector bool char, vector signed char or vector unsigned char when followed by c, d, i, o, u, x or X A vector float, when followed by f.
vh hv	AltiVec: A vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X
vl lv	AltiVec: A vector int, vector unsigned int or vector bool int when followed by c, d, i, o, u, x or X
Flags	
-	The conversion will be left justified.
+	The conversion, if numeric, will be prefixed with a sign (+ or -). By default, only negative numeric values are prefixed with a minus sign (-).

Table 34.5 Length Modifiers And Conversion Specifiers For Formatted Output Functions

space	If the first character of the conversion is not a sign character, it is prefixed with a space. Because the plus sign flag character (+) always prefixes a numeric value with a sign, the space flag has no effect when combined with the plus flag.
#	For c, d, i, and u conversion types, the # flag has no effect. For s conversion types, a pointer to a Pascal string, is output as a character string. For o conversion types, the # flag prefixes the conversion with a 0. For x conversion types with this flag, the conversion is prefixed with a 0x. For e, E, f, g, and G conversions, the # flag forces a decimal point in the output. For g and G conversions with this flag, trailing zeroes after the decimal point are not removed.
0	This flag pads zeroes on the left of the conversion. It applies to d, i, o, u, x, X, e, E, f, g, and G conversion types. The leading zeroes follow sign and base indication characters, replacing what would normally be space characters. The minus sign flag character overrides the 0 flag character. The 0 flag is ignored when used with a precision width for d, i , o, u, x, and X conversion types.
@	Altivec This flag indicates a pointer to a string specified by an argument. This string will be used as a separator for vector elements.

Conversions

d	The corresponding argument is converted to a signed decimal.
i	The corresponding argument is converted to a signed decimal.
o	The argument is converted to an unsigned octal.
u	The argument is converted to an unsigned decimal.
x, X	The argument is converted to an unsigned hexadecimal. The x conversion type uses lowercase letters (abcdef) while X uses uppercase letters (ABCDEF).
n	This conversion type stores the number of items output by printf() so far. Its corresponding argument must be a pointer to an int.

Table 34.5 Length Modifiers And Conversion Specifiers For Formatted Output Functions

f, F	The corresponding floating point argument (<code>float</code> , or <code>double</code>) is printed in decimal notation. The default precision is 6 (6 digits after the decimal point). If the precision width is explicitly 0, the decimal point is not printed. For the <code>f</code> conversion specifier, a double argument representing infinity produces <code>[-]inf</code> ; a double argument representing a NaN (Not a number) produces <code>[-]nan</code> . For the <code>F</code> conversion specifier, <code>[-]INF</code> or <code>[-]NAN</code> are produced instead.
e, E	The floating point argument (<code>float</code> or <code>double</code>) is output in scientific notation: <code>[-]b.aaa±Eee</code> . There is one digit (<code>b</code>) before the decimal point. Unless indicated by an optional precision width, the default is 6 digits after the decimal point (<code>aaa</code>). If the precision width is 0, no decimal point is output. The exponent (<code>ee</code>) is at least 2 digits long. The <code>e</code> conversion type uses lowercase <code>e</code> as the exponent prefix. The <code>E</code> conversion type uses uppercase <code>E</code> as the exponent prefix.
g, G	The <code>g</code> conversion type uses the <code>f</code> or <code>e</code> conversion types and the <code>G</code> conversion type uses the <code>F</code> or <code>E</code> conversion types. Conversion type <code>e</code> (or <code>E</code>) is used only if the converted exponent is less than -4 or greater than the precision width. The precision width indicates the number of significant digits. No decimal point is output if there are no digits following it.
c	The corresponding argument is output as a character.
s	The corresponding argument, a pointer to a character array, is output as a character string. Character string output is completed when a null character is reached. The null character is not output.
p	The corresponding argument is taken to be a pointer. The argument is output using the <code>x</code> conversion type format.
CodeWarrior Extensions	
#s	The corresponding argument, a pointer to a Pascal string, is output as a character string. A Pascal character string is a length byte followed by the number characters specified in the length byte. Note: This conversion type is an extension to the ANSI C library but applied in the same manner as for other format variations.

`printf()`, like `fprintf()`, `sprintf()`, `vfprintf()`, and `vprintf()`, returns the number of arguments that were successfully output. `printf()` returns a negative value if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 342](#)
- [“fprintf” on page 365](#)
- [“sprintf” on page 431](#)
- [“vfprintf” on page 440](#)
- [“vprintf” on page 445](#)
- [“vsprintf” on page 449](#)

Listing 34.23 Example of printf() usage.

```
#include <stdio.h>

int main(void)
{
    int i = 25;
    char c = 'M';
    short int d = 'm';
    static char s[] = "Metrowerks!";
    static char pas[] = "\pMetrowerks again!";
    float f = 49.95;
    double x = 1038.11005;
    int count;
    printf("%s printf() demonstration:\n%n", s, &count);
    printf("The last line contained %d characters\n", count);
    printf("Pascal string output: %#20s\n", pas);
    printf("%-4d %x %06x %-5o\n", i, i, i, i);
    printf("%*d\n", 5, i);
    printf("%4c %4u %4.10d\n", c, c, c);
    printf("%4c %4hu %3.10hd\n", d, d, d);
    printf("$%5.2f\n", f);
    printf("%5.2f\n%6.3f\n%7.4f\n", x, x, x);
    printf("%.*f\n", 8, 5, x);

    return 0;
}
```

The output is:

```
Metrowerks! printf() demonstration:
The last line contained 36 characters
Pascal string output:      Metrowerks again!
25   19 000019 31
25
M   77 0000000077
m   109 0000000109
$49.95
1038.11
1038.110
1038.1101
1038.11005
```

Listing 34.24 Example of AltiVec Printf Extensions

```
#include <stdio.h>

int main(void)
{
    vector signed char s =
        (vector signed char)(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);
    vector unsigned short us16 =
        (vector unsigned short)('a','b','c','d','e','f','g','h');
    vector signed int sv32 =
        (vector signed int)(100, 2000, 30000, 4);
    vector signed int vs32 =
        (vector signed int)(0, -1, 2, 3);
    vector float flt32 =
        (vector float)(1.1, 2.22, 3.3, 4.444);

    printf("s = %vd\n", s);

    printf("s = %,vd\n", s);

    printf("vector=%@vd\n", "\nvector=", s);

    // c specifier so no space is added.
    printf("us16 = %vhc\n", us16);

    printf("sv32 = %,5lvd\n", sv32);

    printf("vs32 = 0x%@.8l vX\n", " , 0x", vs32);

    printf("flt32 = %,.2vf\n", flt32);

    return 0;
}
```

The Result is:

```
s = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
s = 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
vector=1
vector=2
vector=3
vector=4
vector=5
vector=6
vector=7
```

```
vector=8
vector=9
vector=10
vector=11
vector=12
vector=13
vector=14
vector=15
vector=16
us16 = abcdefgh
sv32 = 100, 2000, 30000,      4
vs32 = 0x00000000, 0xFFFFFFFF, 0x00000002, 0x00000003
flt32 = 1.10, 2.22, 3.30, 4.44
```

putc

Write a character to a stream.

```
#include <stdio.h>
int putc(int c, FILE *stream);
```

c	int	The character to write to a file
stream	FILE *	A pointer to a FILE stream

Remarks

The `putc()` function outputs `c` to `stream` and advances `stream`'s file position indicator.

The `putc()` works identically to the `fputc()` function, except that it is written as a macro.

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the `fflush()` function or one of the file positioning operations (`fseek()`, `fsetpos()`, or `rewind()`).

`putc()` returns the character written when successful and return EOF when it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)
[“fputc” on page 371](#)
[“putchar” on page 412](#)

Listing 34.25 Example of putc() usage.

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

int main(void)
{
    FILE *f;
    static char filename[] = "checkputc";
    static char test[] = "flying fish and quail eggs";
    int i;

    // create a new file for output
    if (( f = fopen(filename, "w")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    // output the test character array
    // one character at a time using putc()
    for (i = 0; test[i] > 0; i++)
        putc(test[i], f);

    // close the file
    fclose(f);

    return 0;
}
```

```
Output to file checkputc
flying fish and quail eggs
```

putchar

Write a character to stdout.

```
#include <stdio.h>
int putchar(int c);
```

c	int	The character to write to stdout
---	-----	----------------------------------

Remarks

The putchar() function writes character c to stdout.

putchar() returns c if it is successful and returns EOF if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fputc” on page 371](#)

[“putc” on page 410](#)

Listing 34.26 Example of putchar() usage.

```
#include <stdio.h>

int main(void)
{
    static char test[] = "running jumping walking tree\n";
    int i;

    // output the test character one character
    // at a time until the null character is found.
    for (i = 0; test[i] != '\0'; i++)
        putchar(test[i]);

    return 0;
}
```

Output:

```
running jumping walking tree
```

puts

Write a character string to stdout.

```
#include <stdio.h>

int puts(const char *s);
```

s	const char *	The string written to stdout
----------	--------------	------------------------------

Remarks

The `puts()` function writes a character string array to `stdout`, stopping at, but not including the terminating null character. The function also appends a newline ('`\n`') to the output.

`puts()` returns zero if successful and returns a nonzero value if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

stdio.h

Standard input/output

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fputs” on page 373](#)

Listing 34.27 Example of puts() usage.

```
#include <stdio.h>

int main(void)
{
    static char s[] = "car bus metro werks";
    int i;

    // output the string 10 times
    for (i = 0; i < 10; i++)
        puts(s);

    return 0;
}
```

Output:

```
car bus metro werks
```

remove

Delete a file.

```
#include <stdio.h>

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

filename	const char *	The name of the file to be deleted
----------	--------------	------------------------------------

Remarks

The `remove()` function deletes the named file specified by `filename`.

`remove()` returns 0 if the file deletion is successful, and returns a nonzero value if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“`wremove`” on page 454](#)

[“`fopen`” on page 361](#)

[“`rename`” on page 416](#)

Listing 34.28 Example of `remove()` usage.

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

int main(void)
{
    char filename[40];

    // get a filename from the user
    printf("Enter the name of the file to delete.\n");
    gets(filename);

    // delete the file
    if (remove(filename) != 0) {
        printf("Can't remove %s.\n", filename);
        exit(1);
    }

    return 0;
}
```

rename

Change the name of a file.

```
#include <stdio.h>
int rename(const char *old, const char *new);
```

old	const char *	The old file name
new	const char *	The new file name

Remarks

The `rename()` function changes the name of a file, specified by `old` to the name specified by `new`.

`rename()` returns a nonzero if it fails and returns zero if successful

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“`wrename`” on page 455](#)

[“`freopen`” on page 378](#)

[“`remove`” on page 414](#)

Listing 34.29 Example of rename() usage.

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

int main(void)
{
    char oldname[50];      // current filename
    char newname[50];      // new filename

    // get the current filename from the user
    printf("Please enter the current filename.\n");
    gets(oldname);

    // get the new filename from the user
    printf("Please enter the new filename.\n");
    gets(newname);

    // rename oldname to newname
    if (rename(oldname, newname) != 0) {
        printf("Can't rename %s to %s.\n", oldname,
               newname);
        exit(1);
    }

    return 0;
}
```

Output:

```
Please enter the current filename.
metrowerks
Please enter the new filename.
itwerks
```

rewind

Reset the file position indicator to the beginning of the file.

```
#include <stdio.h>
void rewind(FILE *stream);
```

stream	FILE *	A pointer to a FILE stream
--------	--------	----------------------------

Remarks

The `rewind()` function sets the file indicator position of `stream` such that the next write or read operation will be from the beginning of the file. It also undoes any previous call to `ungetc()` and clears `stream`'s end-of-file and error status.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fseek” on page 387](#)

[“fsetpos” on page 390](#)

Listing 34.30 Example of rewind() usage.

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

int main(void)
{
    FILE *f;
    char filename[80], buf[80];

    // get a filename from the user
    printf("Enter a filename to read.\n");
    gets(filename);

    // open a file for input
    if ((f = fopen(filename, "r")) == NULL) {
        printf("Can't open %s.\n", filename);
        exit(1);
    }

    printf("Reading first line twice.\n");

    // move the file position indicator to the beginning
    // of the file
    rewind(f);
    // read the first line
    fgets(buf, 80, f);
    printf("Once: %s\n", buf);

    // move the file position indicator to the
    //beginning of the file
    rewind(f);

    // read the first line again
    fgets(buf, 80, f);
    printf("Twice: %s\n", buf);

    // close the file
    fclose(f);

    return 0;
}
```

Output:
Enter a filename to read.
itwerks

Reading first line twice.
Once: flying fish and quail eggs
Twice: flying fish and quail eggs

scanf

Read formatted text.

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

format	const char *	The format string
--------	--------------	-------------------

Remarks

The `scanf()` function reads text and converts the text read to programmer specified types.

Scanf Control String and Conversion Specifiers

The `format` argument is a character array containing normal text, white space (space, tab, newline), and conversion specifications. The normal text specifies literal characters that must be matched in the input stream. A white space character indicates that white space characters are skipped until a non-white space character is reached. The conversion specifications indicate what characters in the input stream are to be converted and stored.

The conversion specifications must have matching arguments in the order they appear in `format`. Because `scanf()` stores data in memory, the matching conversion specification arguments must be pointers to objects of the relevant types.

A conversion specification consists of the percent sign (%) prefix, followed by an optional maximum width or assignment suppression, and ending with a conversion type. A percent sign can be skipped by doubling it in `format`; %% signifies a single % in the input stream.

An optional width is a decimal number specifying the maximum width of an input field. `scanf()` will not read more characters for a conversion than is specified by the width.

An optional assignment suppression character (*) can be used to skip an item by reading it but not assigning it. A conversion specification with assignment suppression must not have a corresponding argument.

The last character, the conversion type, specifies the kind of conversion requested. [“Length Modifiers And Conversion Specifiers For Formatted Input Functions” on page 421](#), describes the conversion type characters.

MSL AltiVec Extensions for `scanf`

The AltiVec extensions to the standard `scanf` family of functions is supported in Metrowerks Standard Libraries.

Separator arguments after % and before any specifier may be any character or may be the @ symbol. The @ symbol is a non-Motorola extension that will use a specified string as a specifier.

In the specific case of 'a'c' specifier any char may be used as a separator. For all other specifiers '-' , '+' , '#' , '' may not be used.

The listing [“Example of AltiVec Scanf Extensions” on page 425](#) demonstrates their use.

Table 34.6 Length Modifiers And Conversion Specifiers For Formatted Input Functions

Modifier	Description
Length Specifiers	
hh	The hh flag indicates that the following d, i, o, u, x, X or n conversion specifier applies to an argument that is of type char or unsigned char.
h	The h flag indicates that the following d, i, o, u, x, X or n conversion specifier applies to an argument that is of type short int or unsigned short int.
l	When used with integer conversion specifier, the l flag indicates long int or an unsigned long int type. When used with floating point conversion specifier, the l flag indicates a double. When used with a c or s conversion specifier, the l flag indicates that the corresponding argument with type pointer to wchar_t .

Table 34.6 Length Modifiers And Conversion Specifiers For Formatted Input Functions

ll	When used with integer conversion specifier, the ll flag indicates that the corresponding argument is of type long long or an unsigned long long.
L	The L flag indicates that the corresponding float conversion specifier corresponds to an argument of type long double.
v	AltiVec: A vector bool char, vector signed char or vector unsigned char when followed by c, d, i, o, u, x or X A vector float, when followed by f.
vh hv	AltiVec: vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X
vl lv	AltiVec: vector long, vector unsigned long or vector bool when followed by c, d, i, o, u, x or X

Conversion Specifiers

d	A decimal integer is read.
i	A decimal, octal, or hexadecimal integer is read. The integer can be prefixed with a plus or minus sign (+, -), 0 for octal numbers, 0x or 0X for hexadecimal numbers.
o	An octal integer is read.
u	An unsigned decimal integer is read.
x, X	A hexadecimal integer is read.
e, E, f, g, G	A floating point number is read. The number can be in plain decimal format (e.g. 3456.483) or in scientific notation ([-] b.aaaae [-] dd)
s	A character string is read. The input character string is considered terminated when a white space character is reached or the maximum width has been reached. The null character is appended to the end of the array.
c	A character is read. White space characters are not skipped, but read using this conversion specifier.
p	A pointer address is read. The input format should be the same as that output by the p conversion type in printf().

Table 34.6 Length Modifiers And Conversion Specifiers For Formatted Input Functions

n	This conversion type does not read from the input stream but stores the number of characters read in its corresponding argument.
[scanfset]	Input stream characters are read and filtered determined by the scanfset. See " Scanset " on page 383, for a full description.

`scanf()` returns the number of items successfully read and returns EOF if a conversion type does not match its argument or and end-of-file is reached.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fscanf” on page 379](#)

[“sscanf” on page 432](#)

stdio.h*Standard input/output*

Listing 34.31 Example of scanf() usage.

```
#include <stdio.h>

int main(void)
{
    int i;
    unsigned int j;
    char c;
    char s[40];
    double x;

    printf("Enter an integer surrounded by ! marks\n");
    scanf("!%d!", &i);
    printf("Enter three integers\n");
    printf("in hexadecimal, octal, or decimal.\n");
    // note that 3 integers are read, but only the last two
    // are assigned to i and j
    scanf("%*i %i %ui", &i, &j);

    printf("Enter a character and a character string.\n");
    scanf("%c %10s", &c, s);

    printf("Enter a floating point value.\n");
    scanf("%lf", &x);

    return 0;
}
```

Output:

```
Enter an integer surrounded by ! marks
!94!
Enter three integers
in hexadecimal, octal, or decimal.
1A 6 24
Enter a character and a character string.
Enter a floating point value.
A
Sounds like 'works'!
3.4
```

Listing 34.32 Example of AltiVec Scanf Extensions

```
#include <stdio.h>

int main(void)
{
    vector signed char v8, vs8;
    vector unsigned short v16;
    vector signed long v32;
    vector float vf32;

    sscanf("1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16", "%vd", &v8);
    sscanf("1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16", "%,vd", &vs8);
    sscanf("abcdefg", "%vhc", &v16);
    sscanf("1, 4, 300, 400", "%,3lvd", &v32);
    sscanf("1.10, 2.22, 3.333, 4.4444", "%,5vf", &vf32);

    return 0;
}
```

The Result is:

```
v8 = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16;
vs8 = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16;
v16 = 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'
v32 = 1, 4, 300, 400
vf32 = 1.1000, 2.2200, 3.3330, 4.4444
```

setbuf

Change the buffer size of a stream.

```
#include <stdio.h>

void setbuf(FILE *stream, char *buf);
```

stream	FILE *	A pointer to a FILE stream
buf	char *	A buffer for input or output

Remarks

The `setbuf()` function allows the programmer to set the buffer size for `stream`. It should be called after `stream` is opened, but before it is read from or written to.

The function makes the array pointed to by `buf` the buffer used by `stream`. The `buf` argument can either be a null pointer or point to an array of size `BUFSIZ` defined in `stdio.h`.

If `buf` is a null pointer, the stream becomes unbuffered.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“setvbuf” on page 428](#)

[“malloc” on page 484](#)

Listing 34.33 Example of setbuf() usage.

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

int main(void)
{
    FILE *f;
    char name[80];

    // get a filename from the user
    printf("Enter the name of the file to write to.\n");
    gets(name);

    // create a new file for output
    if ( (f = fopen(name, "w")) == NULL) {
        printf("Can't open file %s.\n", name);
        exit(1);
    }

    setbuf(f, NULL);      // turn off buffering

    // this text is sent directly to the file without
    // buffering
    fprintf(f, "Buffering is now off\n");
    fprintf(f, "for this file.\n");

    // close the file
    fclose(f);

    return 0;
}
```

Output:
Enter the name of the file to write to.
bufftest

setvbuf

Change the buffering scheme for a stream.

```
#include <stdio.h>

int setvbuf(FILE *stream, char *buf, int mode,
             size_t size);
```

stream	FILE *	A pointer to a FILE stream
buf	char *	A buffer for input and output
mode	int	A buffering mode
size	size_t	The size of the buffer

Remarks

The `setvbuf()` allows the manipulation of the buffering scheme as well as the size of the buffer used by `stream`. The function should be called after the stream is opened but before it is written to or read from.

The `buf` argument is a pointer to a character array. The `size` argument indicates the size of the character array pointed to by `buf`. The most efficient buffer size is a multiple of `BUFSIZ`, defined in `stdio.h`.

If `buf` is a null pointer, then the operating system creates its own buffer of `size` bytes.

The `mode` argument specifies the buffering scheme to be used with `stream`. `mode` can have one of three values defined in `stdio.h`: `_IOFBF`, `_IOLBF`, and `_IONBF`.

- `_IOFBF` specifies that `stream` be buffered.
- `_IOLBF` specifies that `stream` be line buffered.
- `_IONBF` specifies that `stream` be unbuffered

`setvbuf()` returns zero if it is successful and returns a nonzero value if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“setbuf” on page 425](#)

[“malloc” on page 484](#)

Listing 34.34 Example of setvbuf() usage.

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

int main(void)
{
    FILE *f;
    char name[80];

    // get a filename from the user
    printf("Enter the name of the file to write to.\n");
    gets(name);

    // create a new file for output
    if ( (f = fopen(name, "w")) == NULL) {
        printf("Can't open file %s.\n", name);
        exit(1);
    }

    setvbuf(f, NULL, _IOLBF, 0);    // line buffering
    fprintf(f, "This file is now\n");
    fprintf(f, "line buffered.\n");

    // close the file
    fclose(f);

    return 0;
}
```

Output:
Enter the name of the file to write to.
buffy

snprintf

Format a character string array.

```
#include <stdio.h>
int snprintf(char * s, size_t n, const char * format, ...);
```

s	char *	A string to write to
n	size_t	Max number of chars to be written to s
format	const char *	The format string

Remarks

The `snprintf()` function works identically to `fprintf()` except that the output is written into the array `s` instead of to a stream. If `n` is zero nothing is written; otherwise, any characters beyond the `n-1`st are discarded rather than being written to the array and a `null` character is appended at the end.

For specifications concerning the output control string and conversion specifiers please see: [“Output Control String and Conversion Specifiers” on page 365](#).

`Snprintf()` returns the number of characters that would have been assigned to `s`, had `n` been sufficiently large, not including the `null` character or a negative value if an encoding error occurred. Thus, the null-terminated output will have been completely written if and only if the returned value is nonnegative and less than `n`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 34.35 Example Of Snprintf() Usage

```
#include <stdio.h>

int main()
{
    int i = 1;
    static char s[] = "Metrowerks";
    char dest[50];
    int retval;

    retval = snprintf(dest, 5, "%s is number %d!", s, i);
    printf("n too small, dest = |%s|, retval = %i\n", dest, retval);
    retval = snprintf(dest, retval, "%s is number %d!", s, i);
    printf("n right size, dest = |%s|, retval = %i\n", dest, retval);

    return 0;
}
```

Output:

```
n too small, dest = |Metr|, retval = 23
n right size, dest = |Metrowerks is number 1|, retval = 23
```

sprintf

Format a character string array.

```
#include <stdio.h>

int sprintf(char *s, const char *format, ...);
```

s	char *	A string to write to
format	const char *	The format string

Remarks

The `sprintf()` function works identically to `printf()` with the addition of the `s` parameter. Output is stored in the character array pointed to by `s` instead of being sent to `stdout`. The function terminates the output character string with a null character.

stdio.h*Standard input/output*

For specifications concerning the output control string and conversion specifiers please see: [“Output Control String and Conversion Specifiers” on page 365](#).

`sprintf()` returns the number of characters assigned to `s`, not including the null character.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fprintf” on page 365](#)

[“printf” on page 402](#)

Listing 34.36 Example of `sprintf()` usage.

```
#include <stdio.h>

int main(void)
{
    int i = 1;
    static char s[] = "Metrowerks";
    char dest[50];

    sprintf(dest, "%s is number %d!", s, i);
    puts(dest);

    return 0;
}
```

Output:

Metrowerks is number 1!

sscanf

Read formatted text into a character string.

```
#include <stdio.h>

int sscanf(char *s, const char *format, ...);
```

s	char *	The string to be scanned
format	const char *	The format string

Remarks

The `sscanf()` operates identically to `scanf()` but reads its input from the character array pointed to by `s` instead of `stdin`. The character array pointed to `s` must be null terminated.

For specifications concerning the input control string and conversion specifications see: “[Input Control String and Conversion Specifiers” on page 380](#). Also see “[Scanset” on page 383](#), for a full description of the use of scansets.

`scanf()` returns the number of items successfully read and converted and returns `EOF` if it reaches the end of the string or a conversion specification does not match its argument.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fscanf” on page 379](#)

[“scanf” on page 420](#)

stdio.h

Standard input/output

Listing 34.37 Example of sscanf() usage.

```
#include <stdio.h>

int main(void)
{
    static char in[] = "figs cat pear 394 road 16!";
    char s1[20], s2[20], s3[20];
    int i;

    // get the words figs, cat, road,
    // and the integer 16
    // from in and store them in s1, s2, s3, and i,
    // respectively
    sscanf(in, "%s %s pear %d", s1, s2, &i);
    printf("%s %s %d\n", s1, s2, i);

    return 0;
}
```

Output:

```
figs cat road 16
```

tmpfile

Open a temporary file.

```
#include <stdio.h>
FILE *tmpfile(void);
```

Remarks

The `tmpfile()` function creates and opens a binary file that is automatically removed when it is closed or when the program terminates.

`tmpfile()` returns a pointer to the `FILE` variable of the temporary file if it is successful. If it fails, `tmpfile()` returns a null pointer (`NULL`).

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“fopen” on page 361](#)
[“tmpnam” on page 435](#)

Listing 34.38 Example of tmpfile() usage.

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

int main(void)
{
    FILE *f;

    // create a new temporary file for output
    if ( (f = tmpfile()) == NULL) {
        printf("Can't open temporary file.\n");
        exit(1);
    }

    // output text to the temporary file
    fprintf(f, "watch clock timer glue\n");

    // close AND DELETE the temporary file
    // using fclose()
    fclose(f);

    return 0;
}
```

tmpnam

Create a unique temporary filename.

```
#include <stdio.h>
char *tmpnam(char *s);
```

s	char *	A temporary file name
---	--------	-----------------------

Remarks

The `tmpnam()` function creates a valid filename character string that will not conflict with any existing filename. A program can call the function up to `TMP_MAX` times before exhausting the unique filenames `tmpnam()` generates. The `TMP_MAX` macro is defined in `stdio.h`.

The `s` argument can either be a null pointer or pointer to a character array. The character array must be at least `L_tmpnam` characters long. The new temporary filename is placed in this array. The `L_tmpnam` macro is defined in `stdio.h`.

If `s` is `NULL`, `tmpnam()` returns with a pointer to an internal static object that can be modified by the calling program.

Unlike `tmpfile()`, a file created using a filename generated by the `tmpnam()` function is not automatically removed when it is closed.

`tmpnam()` returns a pointer to a character array containing a unique, non-conflicting filename. If `s` is a null pointer (`NULL`), the pointer refers to an internal static object. If `s` points to a character array, `tmpnam()` returns the same pointer.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fopen” on page 361](#)

[“tmpfile” on page 434](#)

Listing 34.39 Example of tmpnam() usage.

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

int main(void)
{
    FILE *f;
    char *tempname;
    int c;

    // get a unique filename
    tempname = tmpnam("tempwerks");

    // create a new file for output
    if ( (f = fopen(tempname, "w")) == NULL) {
        printf("Can't open temporary file %s.\n", tempname);
        exit(1);
    }

    // output text to the file
    fprintf(f, "shoe shirt tie trousers\n");
    fprintf(f, "province\n");

    // close the file
    fclose(f);

    // delete the file
    remove(tempname);

    return 0;
}
```

ungetc

Place a character back into a stream.

```
#include <stdio.h>

int ungetc(int c, FILE *stream);
```

c	int	The character to return to a file
stream	FILE *	A pointer to a FILE stream

Remarks

The `ungetc()` function places character `c` back into `stream`'s buffer. The next read operation will read the character placed by `ungetc()`. Only one character can be pushed back into a buffer until a read operation is performed.

The function's effect is ignored when an `fseek()`, `fsetpos()`, or `rewind()` operation is performed.

`ungetc()` returns `c` if it is successful and returns `EOF` if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fseek” on page 387](#)

[“fsetpos” on page 390](#)

[“rewind” on page 417](#)

Listing 34.40 Example of ungetc() usage.

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

int main(void)
{
    FILE *f;
    int c;

    // create a new file for output and input
    if ( (f = fopen("myfoo", "w+")) == NULL) {
        printf("Can't open myfoo.\n");
        exit(1);
    }

    // output text to the file
    fprintf(f, "The quick brown fox\n");
    fprintf(f, "jumped over the moon.\n");

    // move the file position indicator
    // to the beginning of the file
    rewind(f);

    printf("Reading each character twice.\n");

    // read a character
    while ( (c = fgetc(f)) != EOF) {
        putchar(c);
        // put the character back into the stream
        ungetc(c, f);
        c = fgetc(f); // read the same character again
        putchar(c);
    }

    fclose(f);

    return 0;
}
```

Output

```
Reading each character twice.
TThhee qquuiicckk bbrroowwnn ffooxx
jjuuummppeedd oovveerr tthhee mmoooonn..
```

vfprintf

Write formatted output to a stream.

```
#include <stdarg.h>
#include <stdio.h>
int vfprintf(FILE *stream, const char *format, va_list arg);
```

stream	FILE *	A pointer to a FILE stream
format	const char *	The format string
arg	va_list	The variable argument list

Remarks

The vfprintf() function works identically to the fprintf() function. Instead of the variable list of arguments that can be passed to fprintf(), vfprintf() accepts its arguments in the array `arg` of type `va_list` which must have been initialized by the `va_start()` macro from the `stdarg.h` header file. The vfprintf() does not invoke the `va_end` macro.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

For specifications concerning the output control string and conversion specifiers please see: [“Output Control String and Conversion Specifiers” on page 365](#).

`vfprintf()` returns the number of characters written or `EOF` if it failed.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fprintf” on page 365](#)

[“printf” on page 402](#)

[“Overview of stdarg.h” on page 317](#)

Listing 34.41 Example of vfprintf() usage.

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

int fpr(FILE *, char *, ...);

int main(void)
{
    FILE *f;
    static char name[] = "foo";
    int a = 56, result;
    double x = 483.582;

    // create a new file for output
    if (( f = fopen(name, "w")) == NULL) {
        printf("Can't open %s.\n", name);
        exit(1);
    }

    // format and output a variable number of arguments
    // to the file
    result = fpr(f, "%10s %4.4f %-10d\n", name, x, a);

    // close the file
    fclose(f);

    return 0;
}

// fpr() formats and outputs a variable
// number of arguments to a stream using
// the vfprintf() function
int fpr(FILE *stream, char *format, ...)
{
    va_list args;
    int retval;

    va_start(args, format);      // prepare the arguments
    retval = vfprintf(stream, format, args);
    // output them
    va_end(args);              // clean the stack
    return retval;
}
```

stdio.h*Standard input/output*

```
Output to file foo:  
foo 483.5820 56
```

vfscanf

Read formatted text from a stream.

```
#include <stdarg.h>  
  
#include <stdio.h>  
  
int vfscanf(FILE *stream, const char *format, va_list arg);
```

stream	FILE *	A pointer to a FILE stream
format	const char *	The format string
arg	va_list	The variable argument list

Remarks

The vfscanf() function works identically to the fscanf() function. Instead of the variable list of arguments that can be passed to fscanf(), vfscanf() accepts its arguments in the array arg of type va_list, which must have been initialized by the va_start() macro from the stdarg.h header file. The vfscanf() does not invoke the va_end macro.

On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

For specifications concerning the output control string and conversion specifiers please see: [“Length Modifiers And Conversion Specifiers For Formatted Input” on page 382.](#)

vfscanf() returns the number of items assigned, which can be fewer than provided for in the case of an early matching failure. If an input failure occurs before any conversion, vfscanf() returns EOF.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“scanf” on page 420](#)

[“fscanf” on page 379](#)

Listing 34.42 Example Of vfscanf() Usage

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

int fsc(FILE *, char *, ...);

int main(void)
{
    FILE *f;
    int i;
    double x;
    char c;
    int numassigned;
    // create a new file for output and input
    if (( f = fopen("foobar", "w+")) == NULL) {
        printf("Can't create new file.\n");
        exit(1);
    }
    // output formatted text to the file
    fprintf(f, "%d\n%lf\n%c\n", 45, 983.3923, 'M');
    // go to the beginning of the file
    rewind(f);
    // read from the stream using fscanf()
    numassigned = fsc(f, "%d %lf %c", &i, &x, &c);
    // close the file
    fclose(f);
    printf("The number of assignments is %d.\n", numassigned);
    printf("The integer read is %d.\n", i);
    printf("The floating point value is %f.\n", x);
    printf("The character is %c.\n", c);
    return 0;
}

// fsc() scans an input stream and inputs
// a variable number of arguments using
// the vfscanf() function
int fsc(FILE *stream, char *format, ...)
{
    va_list args;
    int retval;

    va_start(args, format);      // prepare the arguments
    retval = vfscanf(stream, format, args);
    va_end(args);              // clean the stack
```

```
    return retval;
}

Output:
The number of assignments is 3.
The integer read is 45.
The floating point value is 983.392300.
The character is M.
```

vprintf

Write formatted output to stdout.

```
#include <stdio.h>

int vprintf(const char *format, va_list arg);
```

format	const char *	The format string
arg	va_list	A variable argument list

Remarks

The `vprintf()` function works identically to the `printf()` function. Instead of the variable list of arguments that can be passed to `printf()`, `vprintf()` accepts its arguments in the array of type `va_list` processed by the `va_start()` macro from the `stdarg.h` header file.

For specifications concerning the output control string and conversion specifiers please see: [“Output Control String and Conversion Specifiers” on page 365](#).

`vprintf()` returns the number of characters written or a negative value if it failed.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)

[“fprintf” on page 365](#)

[“printf” on page 402](#)

[“Overview of stdarg.h” on page 317](#)

Listing 34.43 Example of vprintf() usage.

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

int pr(char *, ...);

int main(void)
{
    int a = 56;
    double f = 483.582;
    static char s[] = "Metrowerks";

    // output a variable number of arguments to stdout
    pr("%15s %4.4f %-10d*\n", s, f, a);

    return 0;
}

// pr() formats and outputs a variable number of arguments
// to stdout using the vprintf() function
int pr(char *format, ...)
{
    va_list args;
    int retval;
    va_start(args, format); // prepare the arguments
    retval = vprintf(format, args);
    va_end(args);           // clean the stack
    return retval;
}
```

Output:

Metrowerks 483.5820 56 *

vsnprintf

Format a character string array.

```
#include <stdarg.h>
#include <stdio.h>
int vsnprintf(char * s, size_t n,
              const char * format, va_list arg);
```

s	char *	A string to write to
n	size_t	Max number of chars to be written to s
format	const char *	The format string
arg	va_list	A variable argument list

Remarks

The `vsnprintf()` function works identically to `snprintf()`, except that the variable list of arguments that can be passed to `snprintf()` is replaced by an array `arg` of type `va_list`, which must have been initialized by the `va_start()` macro from the `stdarg.h` header file. The `vsnprintf()` does not invoke the `va_end` macro. If `n` is zero nothing is written; otherwise, any characters beyond the `n`-1st are discarded rather than being written to the array and a null character is appended at the end.

For specifications concerning the output control string and conversion specifiers please see: [“Output Control String and Conversion Specifiers” on page 365](#).

`Vsnprintf()` returns the number of characters that would have been assigned to `s`, had `n` been sufficiently large, not including the null character or a negative value if an encoding error occurred. Thus, the null-terminated output will have been completely written if and only if the returned value is nonnegative and less than `n`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 342](#)
[“printf” on page 402](#)

[“sprintf” on page 431](#)

[“Overview of stdarg.h” on page 317](#)

Listing 34.44 Example Of Vsprintf() Usage.

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

int sp(char *, size_t, char *, ...);

int main()
{
    int i = 1;
    static char s[] = "Metrowerks";
    char dest[50];
    int retval;

    retval = sp(dest, 5, "%s is number %d!", s, i);
    printf("n too small, dest = |%s|, retval = %i\n", dest, retval);
    retval = sp(dest, retval, "%s is number %d!", s, i);
    printf("n right size, dest = |%s|, retval = %i\n", dest, retval);

    return 0;
}

// sp() formats and outputs a variable number of arguments
// to a character string using the vsnprintf() function
int sp(char * s, size_t n, char *format,...)
{
    va_list args;
    int retval;

    va_start(args, format);      // prepare the arguments
    retval = vsnprintf(s, n, format, args);
    va_end(args);               // clean the stack
    return retval;
}

Output:
n too small, dest = |Metr|, retval = 23
n right size, dest = |Metrowerks is number 1|, retval = 23
```

vsprintf

Write formatted output to a string.

```
#include <stdio.h>
int vsprintf(char *s,
             const char *format, va_list arg);
```

s	char *	A string to write to
format	const char *	The format string
arg	va_list	A variable argument list

Remarks

The `vsprintf()` function works identically to the `sprintf()` function. Instead of the variable list of arguments that can be passed to `sprintf()`, `vsprintf()` accepts its arguments in the array of type `va_list` processed by the `va_start()` macro from the `stdarg.h` header file.

For specifications concerning the output control string and conversion specifiers please see: [“Output Control String and Conversion Specifiers” on page 365](#).

`vsprintf()` returns the number of characters written to `s` not counting the terminating null character. Otherwise, EOF on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 342](#)
- [“printf” on page 402](#)
- [“sprintf” on page 431](#)
- [“Overview of stdarg.h” on page 317](#)

Listing 34.45 Example of vsprintf() usage.

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

int spr(char *, char *, ...);

int main(void)
{
    int a = 56;
    double x = 1.003;
    static char name[] = "Metrowerks";
    char s[50];

    // format and send a variable number of arguments
    // to character array s
    spr(s, "%10s\n %f\n %-10d\n", name, x, a);
    puts(s);

    return 0;
}

// spr() formats and sends a variable number of
// arguments to a character array using the sprintf()
// function
int spr(char *s, char *format, ...)
{
    va_list args;
    int retval;

    va_start(args, format); // prepare the arguments
    retval = vsprintf(s, format, args);
    va_end(args);           // clean the stack
    return retval;
}
```

Output:

```
Metrowerks
1.003000
56
```

vsscanf

Read formatted text from a character string.

```
#include <stdarg.h>
#include <stdio.h>
int vsscanf(const char * s,
            const char * format, va_list arg);
```

s	char *	The character string to be scanned
format	char *	The format string
arg	va_list	The variable argument list

Remarks

The `vsscanf()` function works identically to the `sscanf()` function. Instead of the variable list of arguments that can be passed to `sscanf()`, `vsscanf()` accepts its arguments in the array `arg` of type `va_list`, which must have been initialized by the `va_start()` macro from the `stdarg.h` header file. The `vfscanf()` does not invoke the `va_end` macro.

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

For specifications concerning the output control string and conversion specifiers please see: [“Length Modifiers And Conversion Specifiers For Formatted Input” on page 382](#).

`vfscanf()` returns the number of items assigned, which can be fewer than provided for in the case of an early matching failure. If an input failure occurs before any conversion, `vfscanf()` returns EOF.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“scanf” on page 420](#)
- [“fscanf” on page 379](#)

stdio.h*Standard input/output*

Listing 34.46 Example Of Vsscanf() Usage

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

int ssc(char *, char *, ...);

int main(void)
{
    static char in[] = "figs cat pear 394 road 16!";
    char s1[20], s2[20], s3[20];
    int i;

    // get the words figs, cat, road,
    // and the integer 16
    // from in and store them in s1, s2, s3, and i,
    // respectively
    ssc(in, "%s %s pear %d", s1, s2, &i);
    printf("%s %s %s %d\n", s1, s2, s3, i);

    return 0;
}

// ssc() scans a character string and inputs
// a variable number of arguments using
// the vsscanf() function
int ssc(char * s, char *format, ...)
{
    va_list args;
    int retval;

    va_start(args, format);      // prepare the arguments
    retval = vsscanf(s, format, args);
    va_end(args);               // clean the stack
    return retval;
}

Output:
figs cat road 16
```

_wfopen

Open a file as a stream with a wide character file name.

```
#include <stdio.h>
FILE *_wfopen(const wchar_t *wfilename, const wchar_t *wmode);
```

wfilename	const wchar_t *	The wide character filename of the file to open
wmode	const wchar_t *	The wide character file opening mode

Remarks

The `_wfopen()` function is a wide character implementation of “[fopen” on page 361](#)

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“freopen” on page 378](#)

[“fopen” on page 361](#)

_wfreopen

Re-direct a stream to another file as a wide character version.

```
#include <stdio.h>
FILE *_wfreopen(const wchar_t *wfilename,
const wchar_t *wmode, FILE *stream);
```

wfilename	const wchar_t *	The wide character name of the file to re-open
-----------	-----------------	--

wmode	const wchar_t *	The wide character file opening mode
stream	FILE *	A pointer to a FILE stream

Remarks

The _wfreopen() function is a wide character implementation of [“fopen” on page 378](#)

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“fopen” on page 378](#)

[“_wfopen” on page 453](#)

_wremove

Delete a file.

```
#include <stdio.h>
int remove(const wchar_t *wfilename);
```

wfilename	const wchar_t *	The name of the wide character file to be deleted
-----------	-----------------	---

Remarks

The _fremove() function is a wide character variation of [“remove” on page 414](#)

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“remove” on page 414](#)

_wrename

Change the name of a file.

```
#include <stdio.h>  
  
int rename(const char *wold, const wchar_t *wnew);
```

wold	const wchar_t *	The old wide character file name
wnew	const wchar_t *	The new wide character file name

Remarks

The `_wrename()` function implements a wide character variation of “[rename](#)” [on page 416](#).

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“rename” on page 416](#)

[“_wtmpnam” on page 455](#)

_wtmpnam

Create a unique temporary filename wide character variant.

```
#include <stdio.h>  
  
wchar_t *_wtmpnam(wchar_t *ws);
```

ws	wchar_t *	A temporary wide character file name
----	-----------	--------------------------------------

Remarks

The `_wtmpnam()` functions creates a valid filename wide character string that will not conflict with any existing filename. It is implemented for a wide character array in the same manner as [“tmpnam” on page 435](#)

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“fopen” on page 361](#)

[“tmpfile” on page 434](#)

stdlib.h

The stdlib.h header file provides groups of closely related functions for string conversion, pseudo-random number generation, memory management, environment communication, searching and sorting, multibyte character conversion, and integer arithmetic.

Overview of stdlib.h

The stdlib.h header file provides groups of closely related functions

- [“String Conversion Functions”](#)
- [“Pseudo-random Number Generation Functions”](#)
- [“Memory Management Functions”](#)
- [“Environment Communication Functions”](#)
- [“Searching And Sorting Functions”](#)
- [“Multibyte Conversion Functions”](#)
- [“Integer Arithmetic Functions”](#)

String Conversion Functions

The string conversion functions are

- [“atof” on page 465](#)
- [“atoi” on page 466](#)
- [“atol” on page 467](#)
- [“atoll” on page 468](#)
- [“strtod” on page 493](#)
- [“strtodf” on page 496](#)
- [“strtold” on page 497](#)
- [“strtold” on page 500](#)

-
- [“strtoul” on page 502](#)
 - [“strtoull” on page 503](#)

Pseudo-random Number Generation Functions

The pseudo-random number generation functions are

- [“rand” on page 489](#)
- [“rand_r” on page 491](#)
- [“srand” on page 492](#)

Memory Management Functions

The memory management functions are

- [“calloc” on page 473](#)
- [“free” on page 479](#)
- [“malloc” on page 484](#)
- [“realloc” on page 491](#)
- [“vec_malloc” on page 505](#)
- [“vec_free” on page 506](#)
- [“vec_realloc” on page 507](#)
- [“vec_calloc” on page 508](#)

Environment Communication Functions

The environment communication functions are

- [“abort” on page 459](#)
- [“atexit” on page 462](#)
- [“exit” on page 477](#)
- [“_Exit” on page 479](#)
- [“getenv” on page 480](#)
- [“_putenv” on page 488](#)
- [“system” on page 505](#)

Searching And Sorting Functions

The searching and sorting functions are

- [“bsearch” on page 469](#)
- [“qsort” on page 488](#)

Multibyte Conversion Functions

The multibyte conversion functions convert UTF-8 multibyte characters to wchar_t type characters (defined in `stddef.h`). The functions are

- [“mblen” on page 485](#)
- [“mbstowcs” on page 486](#)
- [“mbtowc” on page 487](#)
- [“wcstombs” on page 508](#)
- [“wctomb” on page 509](#)

Integer Arithmetic Functions

The integer arithmetic functions are

- [“abs” on page 461](#)
- [“div” on page 476](#)
- [“labs” on page 481](#)
- [“llabs” on page 483](#)
- [“ldiv” on page 482](#)

Many of the `stdlib.h` functions use the `size_t` type as well as the `NULL` and `MB_CUR_MAX` macros, which are defined in `stdlib.h`. The macro `MB_CUR_MAX` defines the maximum number of bytes in a single multibyte character.

abort

Abnormal program termination.

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

Remarks

The `abort()` function raises the SIGABRT signal and quits the program to return to the operating system.

The `abort()` function will not terminate the program if a programmer-installed signal handler uses `longjmp()` instead of returning normally.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“assert” on page 41](#)

[“longjmp” on page 280](#)

[“raise” on page 290](#)

[“signal” on page 287](#)

[“atexit” on page 462](#)

[“exit” on page 477](#)

Listing 35.1 Example of abort() usage.

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

int main(void)
{
    char c;

    printf("Aborting the program.\n");
    printf("Press return.\n");

    // wait for the return key to be pressed
    c = getchar();

    // abort the program
    abort();

    return 0;
}
```

Output:
Aborting the program.
Press return.

abs

Compute the absolute value of an integer.

```
#include <stdlib.h>

int abs(int i);
```

i	int	The value being computed
---	-----	--------------------------

`abs()` returns the absolute value of its argument. Note that the two's complement representation of the smallest negative number has no matching absolute integer representation.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fabs” on page 217](#)

[“labs” on page 481](#)

Listing 35.2 Example of abs() usage.

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

int main(void)
{
    int i = -20;
    long int j = -48323;
    long long k = -9223372036854773307;

    printf("Absolute value of %d is %d.\n", i, abs(i));
    printf("Absolute value of %ld is %ld.\n", j, labs(j));
    printf("Absolute value of %lld is %lld.\n", k, llabs(k));

    return 0;
}
```

Output:

```
Absolute value of -20 is 20.
Absolute value of -48323 is 48323.
Absolute value of -9223372036854773307 is 9223372036854773307.
```

atexit

Install a function to be executed at a program's exit.

```
#include <stdlib.h>

int atexit(void (*func) void));
```

func	void *	The function to execute at exit
------	--------	---------------------------------

Remarks

The `atexit()` function adds the function pointed to by `func` to a list. When `exit()` is called, each function on the list is called in the reverse order in which they were installed with `atexit()`. After all the functions on the list have been called, `exit()` terminates the program.

The `stdio.h` library, for example, installs its own exit function using `atexit()`. This function flushes all buffers and closes all open streams.

`atexit()` returns a zero when it succeeds in installing a new exit function and returns a nonzero value when it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“exit” on page 477](#)

Listing 35.3 Example of atexit() usage.

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

// Prototypes
void first(void);
void second(void);
void third(void);

int main(void)
{
    atexit(first);
    atexit(second);
    atexit(third);

    printf("exiting program\n\n");
    return 0;
}

void first(void)
{
    int c;

    printf("First exit function.\n");
    printf("Press return.\n");
// wait for the return key to be pressed
    c = getchar();
}

void second(void)
{
    int c;

    printf("Second exit function.\n");
    printf("Press return.\n");
    c = getchar();
}

void third(void)
{
    int c;

    printf("Third exit function.\n");
    printf("Press return.\n");
    c = getchar();
```

}

Output:

Third exit function.
Press return.

Second exit function.
Press return.

First exit function.
Press return.

atof

Convert a character string to a numeric value of type double.

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

nptr	const char *	The character being converted
------	--------------	-------------------------------

Remarks

The `atof()` function converts the character array pointed to by `nptr` to a floating point value of type `double`. Except for its behavior on error, this function is the equivalent of the call `strtod(nptr, NULL)`;

This function sets the global variable `errno` to `ERANGE` if the converted value cannot be expressed as a floating point value of type `double`.

`atof()` returns a floating point value of type `double`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“atoi” on page 466](#)
- [“atol” on page 467](#)
- [“errno” on page 95](#)

[“strtod” on page 493](#)[“scanf” on page 420](#)**Listing 35.4 Example of atof(), atoi(), atol() usage.**

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

int main(void)
{
    int i;
    long int j;
    float f;
    static char si[] = "-493", sli[] = "63870";
    static char sf[] = "1823.4034";

    f = atof(sf);
    i = atoi(si);
    j = atol(sli);

    printf("%f %d %ld\n", f, i, j);

    return 0;
}
```

Output:

```
1823.403400 -493 63870
```

atoi

Convert a character string to a value of type int.

```
#include <stdlib.h>

int atoi(const char *nptr);
```

nptr	const char *	The string to be converted
------	--------------	----------------------------

Remarks

The `atoi()` function converts the character array pointed to by `nptr` to an integer value. Except for its behavior on error, this function is the equivalent of the call `(int)strtol(nptr, (char **)NULL, 10);`

This function sets the global variable `errno` to `ERANGE` if the converted value cannot be expressed as a value of type `int`.

`atoi()` returns an integer value of type `int`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“atof” on page 465](#)
- [“atol” on page 467](#)
- [“errno” on page 95](#)
- [“strtol” on page 497](#)
- [“scanf” on page 420](#)

atol

Convert a character string to a value of type `long`.

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

<code>nptr</code>	<code>const char *</code>	The string to be converted
-------------------	---------------------------	----------------------------

Remarks

The `atol()` function converts the character array pointed to by `nptr` to an integer of type `long int`. Except for its behavior on error, this function is the equivalent of the call `strtol(nptr, (char **)NULL, 10);`

This function sets the global variable `errno` to `ERANGE` if the converted value cannot be expressed as a value of type `long int`.

`atol()` returns an integer value of type `long int`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“atof” on page 465](#)
- [“atoi” on page 466](#)
- [“errno” on page 95](#)
- [“strtol” on page 497](#)
- [“scanf” on page 420](#)

atoll

Convert a character string to a value of type long long.

```
#include <stdlib.h>
long long atoll(const char *nptr);
```

nptr	const char *	The string to be converted
------	--------------	----------------------------

Remarks

The `atoll()` function converts the character array pointed to by `nptr` to an integer of type `long long`. Except for its behavior on error, this function is the equivalent of the call `strtoll(nptr, (char **)NULL, 10)`;

This function sets the global variable `errno` to `ERANGE` if the converted value cannot be expressed as a value of type `long int`.

`atoll()` returns an integer value of type `long long`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“atof” on page 465](#)
- [“atoi” on page 466](#)

bsearch

This function uses the binary search algorithm to make an efficient search of a sorted array for an item.

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base,
              size_t num, size_t size,
              int (*compare) (const void *, const void *))
```

key	const void *	Search criteria see remarks
base	const void *	The array to be searched see remarks
num	size_t	Number of elements see remarks
size	size_t	Size of an array element see remarks
compare	const void *	A pointer to a function used for comparison see remarks

Remarks

The `key` argument points to the item you want to search for.

The `base` argument points to the first byte of the array to be searched. This array must already be sorted in ascending order. This order is based on the comparison requirements of the function pointed to by the `compare` argument.

The `num` argument specifies the number of array elements to search.

The `size` argument specifies the size of an array element.

The `compare` argument is a pointer to a programmer-supplied function that is used to compare two elements of the array. That compare function takes two array element pointers as arguments. The first argument is the `key` that was passed to `bsearch()` as the first argument to `bsearch()`. The second argument is a pointer to an element of the array passed as the second argument to `bsearch()`.

For explanation, we will call the arguments `search_key` and `array_element`. The `compare` function compares the `search_key` to the `array_element`. If the `search_key` and the `array_element` are equal, the function will return zero. If the `search_key` is less than the `array_element`, the function will return a negative

value. If the search_key is greater than the array_element, the function will return a positive value.

bsearch() returns a pointer to the element in the array matching the item pointed to by key. If no match was found, bsearch() returns a null pointer (NULL).

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“qsort” on page 488](#)

Listing 35.5 Example of bsearch usage.

```
// A simple telephone directory manager
// This program accepts a list of names and
// telephone numbers, sorts the list, then
// searches for specified names.

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// Maximum number of records in the directory.
#define MAXDIR 40

typedef struct
{
    char lname[15];           // keyfield--see comp() function
    char fname[15];
    char phone[15];
} DIRENTRY;                  // telephone directory record

int comp(const DIRENTRY *, const DIRENTRY *);
DIRENTRY *look(char *);
DIRENTRY directory[MAXDIR];      // the directory itself
int reccount;                   // the number of records entered

int main(void)
{
    DIRENTRY *ptr;
    int lastlen;
    char lookstr[15];

    printf("Telephone directory program.\n");
    printf("Enter blank last name when done.\n");

    reccount = 0;
    ptr = directory;
    do {
        printf("\nLast name: ");
        gets(ptr->lname);
        printf("First name: ");
        gets(ptr->fname);
        printf("Phone number: ");
        gets(ptr->phone);
        if ( (lastlen = strlen(ptr->lname)) > 0) {
            reccount++;
    }
}
```

stdlib.h*Overview of stdlib.h*

```
        ptr++;
    }
} while ( (lastlen > 0) && (reccount < MAXDIR) );

printf("Thank you.  Now sorting. . .\n");

// sort the array using qsort()
qsort(directory, reccount,
       sizeof(directory[0]), (void *)comp);

printf("Enter last name to search for,\n");
printf("blank to quit.\n");
printf("\nLast name: ");
gets(lookstr);

while ( (lastlen = strlen(lookstr)) > 0) {
    ptr = look(lookstr);
    if (ptr != NULL)
        printf("%s, %s: %s\n",
               ptr->lname,
               ptr->fname,
               ptr->phone);
    else    printf("Can't find %s.\n", lookstr);
    printf("\nLast name: ");
    gets(lookstr);
}

printf("Done.\n");

return 0;
}

int comp(const DIRENTRY *rec1, const DIRENTRY *rec2)
{
    return (strcmp((char *)rec1->lname,
                   (char *)rec2->lname));
}

// search through the array using bsearch()
DIRENTRY *look(char k[])
{
    return (DIRENTRY *) bsearch(k, directory, reccount,
sizeof(directory[0]), (void *)comp);
}
```

Output

Telephone directory program.
Enter blank last name when done.

Last name: Mation
First name: Infor
Phone number: 555-1212

Last name: Bell
First name: Alexander
Phone number: 555-1111

Last name: Johnson
First name: Betty
Phone number: 555-1010

Last name:
First name:
Phone number:
Thank you. Now sorting. . .
Enter last name to search for,
blank to quit.

Last name: Mation
Infor, Mation: 555-1212

Last name: Johnson
Johnson, Betty: 555-1010

Last name:
Done.

calloc

Allocate space for a group of objects.

```
#include <stdlib.h>  
  
void *calloc(size_t nmemb, size_t elemsize);
```

nmemb	size_t	Number of elements
elemsize	size_t	The size of the elements

Remarks

The `calloc()` function allocates contiguous space for `nmemb` elements of size `elemsize`. The space is initialized with all bits zero.

`calloc()` returns a pointer to the first byte of the memory area allocated.

`calloc()` returns a null pointer (`NULL`) if no space could be allocated.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“vec_calloc” on page 505](#)

[“malloc” on page 484](#)

[“realloc” on page 491](#)

Listing 35.6 Example of calloc() usage.

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

int main(void)
{
    static char s[] = "Metrowerks compilers";
    char *sptr1, *sptr2, *sptr3;

    // allocate the memory three different ways
    // one: allocate a thirty byte block of
    //     uninitialized memory
    sptr1 = (char *) malloc(30);
    strcpy(sptr1, s);
    printf("Address of sptr1: %p\n", sptr1);

    // two: allocate twenty bytes of unitialized memory
    sptr2 = (char *) malloc(20);
    printf("sptr2 before reallocation: %p\n", sptr2);
    strcpy(sptr2, s);
    // now re-allocate ten extra bytes (for a total of
    // thirty bytes)
    //
    // note that the memory block pointed to by sptr2 is
    // still contiguous after the call to realloc()
    sptr2 = (char *) realloc(sptr2, 30);
    printf("sptr2 after reallocation: %p\n", sptr2);

    // three: allocate thirty bytes of initialized memory
    sptr3 = (char *) calloc(strlen(s), sizeof(char));
    strcpy(sptr3, s);
    printf("Address of sptr3: %p\n", sptr3);

    puts(sptr1);
    puts(sptr2);
    puts(sptr3);

    // release the allocated memory to the heap
    free(sptr1);
    free(sptr2);
    free(sptr3);

    return 0;
}
```

Output:

```
Address of sptr1: 5e5432
sptr2 before reallocation: 5e5452
sptr2 after reallocation: 5e5468
Address of sptr3: 5e5488
Metrowerks compilers
Metrowerks compilers
Metrowerks compilers
```

div

Compute the integer quotient and remainder.

```
#include <stdlib.h>
div_t div(int numer, int denom);
```

numer	int	The numerator
denom	int	The denominator

Remarks

The `div_t` type is defined in `stdlib.h` as

```
typedef struct { int quot,rem; } div_t;
```

`div()` divides `denom` into `numer` and returns the quotient and remainder as a `div_t` type.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fmod” on page 220](#)

[“ldiv” on page 482](#)

[“div_t” on page 91](#)

Listing 35.7 Example of div() usage.

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

int main(void)
{
    div_t result;
    ldiv_t lresult;

    int d = 10, n = 103;
    long int ld = 1000L, ln = 1000005L;

    result = div(n, d);
    lresult = ldiv(ln, ld);

    printf("%d / %d has a quotient of %d\n",
           n, d, result.quot);
    printf("and a remainder of %d\n", result.rem);
    printf("%ld / %ld has a quotient of %ld\n",
           ln, ld, lresult.quot);
    printf("and a remainder of %ld\n", lresult.rem);

    return 0;
}
```

Output:

```
103 / 10 has a quotient of 10
and a remainder of 3
1000005 / 1000 has a quotient of 1000
and a remainder of 5
```

exit

Terminate a program normally.

```
#include <stdlib.h>

void exit(int status);
```

status	int	The exit error value
--------	-----	----------------------

Remarks

The `exit()` function calls every function installed with `atexit()` in the reverse order of their installation, flushes the buffers and closes all open streams, then calls the Toolbox system call `ExitToShell()`.

`exit()` does not return any value to the operating system. The `status` argument is kept to conform to the ANSI C Standard Library specification.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“abort” on page 459](#)

[“atexit” on page 462](#)

Listing 35.8 Example of exit() usage.

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

int main(void)
{
    FILE *f;
    int count;

    // create a new file for output exit on failure
    if (( f = fopen("foofoo", "w") ) == NULL) {
        printf("Can't create file.\n");
        exit(1);
    }

    // output numbers 0 to 9
    for (count = 0; count < 10; count++)
        fprintf(f, "%5d", count);

    // close the file
    fclose(f);

    return 0;
}
```

_Exit

The _Exit function causes normal program termination

```
#include <stdlib.h>
void _Exit(int status);
```

status	int	exit status
--------	-----	-------------

Remarks

This function can not return but a status value is passed back to the calling host in the same manner as in the function [“exit” on page 477](#).

The effects on open buffers is implementation defined.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“abort” on page 459](#)

[“atexit” on page 462](#)

For example usage refer to [“Example of exit\(\) usage.” on page 478](#)

free

Release previously allocated memory to heap.

```
#include <stdlib.h>
void free(void *ptr);
```

ptr	void *	A pointer to the allocated memory
-----	--------	-----------------------------------

Remarks

The free() function releases a previously allocated memory block, pointed to by ptr, to the heap. The ptr argument should hold an address returned by the

memory allocation functions `calloc()`, `malloc()`, or `realloc()`. Once the memory block pointed to by `ptr` has been released, it is no longer valid. The `ptr` variable should not be used to reference memory again until it is assigned a value from the memory allocation functions.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSL/Compatibility>](http://www.metrowerks.com/docs/MSL/Compatibility) for a Compatibility list.

See Also

- [“vec_free” on page 506](#)
- [“calloc” on page 473](#)
- [“malloc” on page 484](#)
- [“realloc” on page 491](#)

For example of `free()` usage refer to “[Example of calloc\(\) usage.” on page 475](#) .

getenv

Environment list access.

```
#include <stdlib.h>
char *getenv(char *name);
```

name	char *	A buffer for the environment list
------	--------	-----------------------------------

Remarks

For Macintosh systems the `getenv()` is an empty function that always returns a null pointer (`NULL`). It is included in the Metrowerks `stdlib.h` header file to conform to the ANSI C Standard Library specification.

`getenv()` returns `NULL` for the Mac. For Windows `getenv()` returns zero on failure or the environmental variable.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSL/Compatibility>](http://www.metrowerks.com/docs/MSL/Compatibility) for a Compatibility list.

See Also

- [“system” on page 505](#)

Listing 35.9 Example of getenv() usage:

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

int main(void)
{
    char *value;
    char *var = "path";

    if( (value = getenv(var)) == NULL)
    { printf("%s is not a environmental variable", var); }
    else
    { printf("%s = %s \n", var, value); }

    return 0;
}
```

Result:

```
path = c:\program files\metrowerks\codewarrior;c:\WINNT\system32
```

labs

Compute long integer absolute value.

```
#include <stdlib.h>

long int labs(long int j);
```

j	long int	The variable to be computed
---	----------	-----------------------------

labs() returns the absolute value of its argument as a value of type long int.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fabs” on page 217](#)

[“abs” on page 461](#)

For example of labs() usage refer to [“Example of abs\(\) usage.” on page 462](#).

ldiv

Compute the long integer quotient and remainder.

```
#include <stdlib.h>

ldiv_t ldiv(long int numer, long int denom);
```

numer	long int	The numerator
denom	long int	The denominator

Remarks

The `ldiv_t` type is defined in `stdlib.h` as

```
typedef struct {

    long int quot, rem;

} ldiv_t;
```

`ldiv()` divides `denom` into `numer` and returns the quotient and remainder as an `ldiv_t` type.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fmod” on page 220](#)

[“div” on page 476](#)

[“ldiv_t” on page 92](#)

[“lldiv_t” on page 92](#)

For example of `ldiv()` usage refer to [“Example of div\(\) usage.” on page 477](#).

llabs

Compute long long integer absolute value.

```
#include <stdlib.h>
long long llabs(long long j);
```

j	long long	The variable to be computed
---	-----------	-----------------------------

Remarks

llabs() returns the absolute value of its argument as a value of type long long.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fabs” on page 217](#)

[“abs” on page 461](#)

For example of llabs() usage refer to [“Example of abs\(\) usage.” on page 462](#).

lldiv

Compute the long long integer quotient and remainder.

```
#include <stdlib.h>
ldiv_t ldiv(long long numer, long long denom);
```

numer	long long	The numerator
denom	long long	The denominator

Remarks

The lldiv_t type is defined in <div_t.h> as

```
typedef struct {
```

```
long long quot, rem;  
} lldiv_t;
```

lldiv() divides denom into numer and returns the quotient and remainder as an lldiv_t type.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ldiv” on page 482](#)

[“fmod” on page 220](#)

[“lldiv_t” on page 92](#)

malloc

Allocate a block of heap memory.

```
#include <stdlib.h>  
void *malloc(size_t size);
```

size	size_t	The size in bytes of the allocation
------	--------	-------------------------------------

Remarks

The malloc() function allocates a block of contiguous heap memory size bytes large. If the argument for malloc() is zero the behavior is unspecified. Dependent upon the system, either a null pointer is returned, or the behavior is as if the size was not zero, except that the returned pointer can not be used to access an object.

malloc() returns a pointer to the first byte of the allocated block if it is successful and return a null pointer if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“vec_malloc” on page 507](#)

[“calloc” on page 473](#)

[“free” on page 479](#)

[“realloc” on page 491](#)

For example of malloc() usage refer to [“Example of calloc\(\) usage.” on page 475.](#)

mblen

Compute the length of an encoded multibyte character, encoded as defined by the `LC_CTYPE` category of the current locale.

```
#include <stdlib.h>  
  
int mblen(const char *s, size_t n);
```

s	const char *	The multibyte array to measure
n	size_t	The maximum size

Remarks

The `mblen()` function returns the length of the multibyte character pointed to by `s`. It examines a maximum of `n` characters.

If `s` is a null pointer, the `mblen` function returns a nonzero or zero value signifying whether multibyte encoding do or do not have state-dependent encoding. If `s` is not a null pointer, the `mblen` function either returns 0 (if `s` points to the null character), or returns the number of bytes that are contained in the multibyte.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Locale specification” on page 191](#)

[“mbtowc” on page 487](#)

mbstowcs

Convert a multibyte character array encoded as defined by the LC_CTYPE category of the current locale to a wchar_t array.

```
#include <stdlib.h>
size_t mbstowcs(wchar_t *pwcs,
                 const char *s, size_t n);
```

pwcs	wchar_t *	The wide character destination
s	const char *s	The multibyte string to convert
n	size_t	The maximum wide characters to convert

Remarks

The Metrowerks implementation of `mbstowcs()` converts a sequence of multibyte characters encoded as defined by the LC_CTYPE category of the current locale from the character array pointed to by `s` and stores not more than `n` of the corresponding Unicode characters into the wide character array pointed to by `pwcs`. No multibyte characters that follow a null character (which is converted into a null wide character) will be examined or converted.

If an invalidly encoded character is encountered, `mbstowcs()` returns the value `(size_t)(-1)`. Otherwise `mbstowcs` returns the number of elements of the array pointed to by `pwcs` modified, not including any terminating null wide character.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“Locale specification” on page 191](#)

[“wcstombs” on page 508](#)

mbtowc

Translate a multibyte character, encoded as defined by the `LC_CTYPE` category of the current locale, to a `wchar_t` type.

```
#include <stdlib.h>

int mbtowc(wchar_t *pwc, const char *s, size_t n);
```

pwc	wchar_t *	The wide character destination
s	const char *s	The string to convert
n	size_t	The maximum wide characters to convert

Remarks

If `s` is not a null pointer, the `mbtowc()` function examines at most `n` bytes starting with the byte pointed to by `s` to determine whether the next multibyte character is a complete and valid encoding of a Unicode character encoded as defined by the `LC_CTYPE` category of the current locale. If so, and `pwc` is not a null pointer, it converts the multibyte character, pointed to by `s`, to a character of type `wchar_t`, pointed to by `pwc`.

`mbtowc()` returns `-1` if `n` is zero and `s` is not a null pointer or if `s` points to an incomplete or invalid multibyte encoding.

`mbtowc()` returns `0` if `s` is a null pointer or `s` points to a null character ('`\0`') .

`mbtowc()` returns the number of bytes of `s` required to form a complete and valid multibyte encoding of the Unicode character.

In no case will the value returned be greater than `n` or the value of the macro `MB_CUR_MAX`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Locale specification” on page 191](#)

[“mblen” on page 485](#)

[“wctomb” on page 509](#)

putenv

This functions lets you enter an argument into the environment list.

```
#include <stdlib.h>
char *_putenv(const char *name);
```

name	const char *	The item to add to the environment list
------	--------------	---

The function `_putenv()` returns NULL on success or minus one on failure to enter the environmental variable.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“getenv” on page 480](#)

[“system” on page 505](#)

qsort

Sort an array.

```
#include <stdlib.h>
void qsort(void *base,size_t nmemb,size_t size,
           int (*compare) (const void *, const void *))
```

base	void *	A pointer to the array to be sorted
nmemb	size_t	The number of elements
size	size_t	The size of the elements
compare	void *	A pointer to a comparison function

Remarks

The `qsort()` function sorts an array using the quicksort algorithm. It sorts the array without displacing it; the array occupies the same memory it had before the call to `qsort()`.

The `base` argument is a pointer to the base of the array to be sorted.

The `nmemb` argument specifies the number of array elements to sort.

The `size` argument specifies the size of an array element.

The `compare` argument is a pointer to a programmer-supplied compare function. The function takes two pointers to different array elements and compares them based on the key. If the two elements are equal, `compare` must return a zero. The `compare` function must return a negative number if the first element is less than the second. Likewise, the function must return a positive number if the first argument is greater than the second.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“bsearch” on page 469](#)

For example of `qsort()` usage refer to [“Example of bsearch usage.” on page 471](#).

rand

Generate a pseudo-random integer value.

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

Remarks

A sequence of calls to the `rand()` function generates and returns a sequence of pseudo-random integer values from 0 to `RAND_MAX`. The `RAND_MAX` macro is defined in `stdlib.h`.

By seeding the random number generator using `srand()`, different random number sequences can be generated with `rand()`.

`rand()` returns a pseudo-random integer value between 0 and `RAND_MAX`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“srand” on page 492](#)

Listing 35.10 Example of rand() usage.

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

int main(void)
{
    int i;
    unsigned int seed;

    for (seed = 1; seed <= 5; seed++) {
        srand(seed);
        printf("First five random numbers for seed %d:\n", seed);
        for (i = 0; i < 5; i++)
            printf("%10d", rand());
        printf("\n\n");           // terminate the line
    }

    return 0;
}
```

Output:

```
First five random numbers for seed 1:
    16838      5758      10113      17515      31051

First five random numbers for seed 2:
    908       22817      10239      12914      25837

First five random numbers for seed 3:
    17747      7107      10365      8312       20622

First five random numbers for seed 4:
    1817       24166      10491      3711      15407

First five random numbers for seed 5:
    18655      8457      10616      31877      10193
```

rand_r

Reentrant function to generate a pseudo-random integer value.

```
#include <stdlib.h>
int rand_r(unsigned int *context);
```

context	unsigned int *	The context seed value
---------	----------------	------------------------

Remarks

The `rand_r()` function provides the same service as [“rand” on page 489](#), yet it also combines the functionality of `srand()` as well. The result of `rand_r()` is equivalent to calling `srand()` with a context seed value, then calling `rand()`. The difference is that for `rand_r()`, the caller provides the storage for the context seed value.

This function may require extra library support.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“rand” on page 489](#)
- [“srand” on page 492](#)

realloc

Change the size of an allocated block of heap memory.

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

ptr	void *	A pointer to an allocated block of memory
size	size_t	The size of memory to reallocate

Remarks

The `realloc()` function changes the size of the memory block pointed to by `ptr` to `size` bytes. The `size` argument can have a value smaller or larger than the current size of the block `ptr` points to. The `ptr` argument should be a value assigned by the memory allocation functions `calloc()` and `malloc()`.

If `size` is 0, the memory block pointed to by `ptr` is released. If `ptr` is a null pointer, `realloc()` allocates `size` bytes.

The old contents of the memory block are preserved in the new block if the new block is larger than the old. If the new block is smaller, the extra bytes are cut from the end of the old block.

`realloc()` returns a pointer to the new block if it is successful and `size` is greater than 0. `realloc()` returns a null pointer if it fails or `size` is 0.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“vec_realloc” on page 508](#)

[“calloc” on page 473](#)

[“free” on page 479](#)

[“malloc” on page 484](#)

For example of `realloc()` usage refer to [“Example of calloc\(\) usage.” on page 475](#).

srand

Sets the seed for the pseudo-random number generator.

```
#include <stdlib.h>
void srand(unsigned int start);
```

seed	unsigned int	A seeding value
------	--------------	-----------------

Remarks

The `srand()` function sets the seed for the pseudo-random number generator to start. Further calls of `srand()` with the same seed value produces the same sequence of random numbers.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“rand” on page 489](#)

For example of `labs()` usage refer to [“Example of rand\(\) usage.” on page 490](#).

strtod

Character array conversion to floating point value of type double.

```
#include <stdlib.h>

double strtod( const char *nptr, char **endptr);
```

nptr	const char *	A Null terminated array to convert
endptr	char **	A pointer to a position in nptr that is follows the converted part.

Remarks

The `strtod()` function converts a character array, pointed to by `nptr`, to a floating point value of type double. The character array can be in either decimal or hexadecimal floating point constant notation (e.g. `103.578`, `1.03578e+02`, or `0x1.9fefef9p+6`).

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to a value of type double.

In other than the “C” locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`strtod()` returns a floating point value of type `double`. If `nptr` cannot be converted to an expressible double value, `strtod()` returns `HUGE_VAL`, defined in `math.h`, and sets `errno` to `ERANGE`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“`strtol`” on page 497](#)

[“`strtoul`” on page 502](#)

[“`errno`” on page 95](#)

[“Integral type limits” on page 189](#)

[“Overview of `math.h`” on page 197](#)

[“`scanf`” on page 420](#)

Listing 35.11 Example of strtod() and strtold() usage.

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

int main(void)
{
    double f;
    long double lf;
    static char sf1[] = "103.578 777";
    static char sf2[] = "1.03578e+02 777";
    static char sf3[] = "0x1.9efef9p+6 777";
    char *endptr;

    f = strtod(sf1, &endptr);
    printf("Value = %f remainder of string = |%s|\n", f, endptr);

    f = strtod(sf2, &endptr);
    printf("Value = %f remainder of string = |%s|\n", f, endptr);

    f = strtod(sf3, &endptr);
    printf("Value = %f remainder of string = |%s|\n", f, endptr);

    lf = strtold(sf1, &endptr);
    printf("Value = %lf remainder of string = |%s|\n", lf, endptr);

    lf = strtold(sf2, &endptr);
    printf("Value = %lf remainder of string = |%s|\n", lf, endptr);

    lf = strtold(sf3, &endptr);
    printf("Value = %lf remainder of string = |%s|\n", lf, endptr);

    return 0;
}
```

Output:

```
Value = 103.578000 remainder of string = | 777 |
Value = 103.578000 remainder of string = | 777 |
Value = 103.748997 remainder of string = | 777 |
Value = 103.578000 remainder of string = | 777 |
Value = 103.578000 remainder of string = | 777 |
Value = 103.748997 remainder of string = | 777 |
```

strtod

Character array conversion to floating point value of type float.

```
#include <stdlib.h>

float strtod( const char *nptr, char **endptr);
```

nptr	const char *	A Null terminated array to convert
endptr	char **	A pointer to a position in nptr that is follows the converted part.

Remarks

The strtod() function converts a character array, pointed to by nptr, to a floating point value of type float. The character array can be in either decimal or hexadecimal floating point constant notation (e.g. 103.578, 1.03578e+02, or 0x1.9efef9p+6).

If the endptr argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by nptr. This position marks the first character that is not convertible to a value of type float.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

strtod() returns a floating point value of type float. If nptr cannot be converted to an expressible float value, strtod() returns HUGE_VAL, defined in math.h, and sets errno to ERANGE.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

["strtol" on page 497](#)

["strtod" on page 493](#)

strtol

Character array conversion to an integral value of type long int.

```
#include <stdlib.h>

long int strtol(const char *nptr, char **endptr, int base);
```

nptr	const char *	A Null terminated array to convert
endptr	char **	A pointer to a position in nptry that is not convertible.
base	int	A numeric base between 2 and 36

Remarks

The `strtol()` function converts a character array, pointed to by `nptr`, expected to represent an integer expressed in radix `base`, to an integer value of type `long int`. If the sequence pointed to by `nptr` is a minus sign, the value resulting from the conversion is negated in the return value.

The `base` argument in `strtol()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then `strtol()` converts the character array based on its format. Character arrays beginning with '0' are assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to a `long int` value.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`strtol()` returns an integer value of type `long int`. If the converted value is less than `LONG_MIN`, `strtol()` returns `LONG_MIN` and sets `errno` to `ERANGE`. If the converted value is greater than `LONG_MAX`, `strtol()` returns `LONG_MAX` and sets `errno` to `ERANGE`. The `LONG_MIN` and `LONG_MAX` macros are defined in `limits.h`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

- [“strtod” on page 493](#)
- [“strtoul” on page 502](#)
- [“errno” on page 95](#)
- [“Integral type limits” on page 189](#)
- [“Overview of math.h” on page 197](#)
- [“scanf” on page 420](#)

Listing 35.12 Example of strtol(), strtoul(), strtoll(), and strtoull() usage

```
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    long int i;
    unsigned long int j;
    long long int lli;
    unsigned long long ull;
    static char si[] = "4733 777";
    static char sb[] = "0x10*****";
    static char sc[] = "66E00M??";
    static char sd[] = "Q0N50M abcd";

    char *endptr;
    i = strtol(si, &endptr, 10);
    printf("%ld remainder of string = |%s|\n", i, endptr);
    i = strtol(si, &endptr, 8);
    printf("%ld remainder of string = |%s|\n", i, endptr);
    j = strtoul(sb, &endptr, 0);
    printf("%lu remainder of string = |%s|\n", j, endptr);
    j = strtoul(sb, &endptr, 16);
    printf("%lu remainder of string = |%s|\n", j, endptr);
    lli = strtoll(sc, &endptr, 36);
    printf("%lld remainder of string = |%s|\n", lli, endptr);
    ull = strtoull(sd, &endptr, 27);
    printf("%llu remainder of string = |%s|\n", ull, endptr);
    return 0;
}
```

Output:

```
4733 remainder of string = | 777|
2523 remainder of string = | 777|
16 remainder of string = |*****|
16 remainder of string = |*****|
373527958 remainder of string = |???
373527958 remainder of string = | abcd|
```

strtold

Character array conversion to floating point value of type long double.

```
#include <stdlib.h>

long double strtold( const char *nptr, char **endptr);
```

nptr	const char *	A Null terminated array to convert
endptr	char **	A pointer to a position in nptr that follows the converted part

Remarks

The `strtold()` function converts a character array, pointed to by `nptr`, to a floating point value of type `long double`. The character array can be in either decimal or hexadecimal floating point constant notation (e.g. `103.578`, `1.03578e+02`, or `0x1.9efef9p+6`).

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to a value of type `double`.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`strtold()` returns a floating point value of type `long double`. If `nptr` cannot be converted to an expressible `double` value, `strtold()` returns `HUGE_VAL`, defined in `math.h`, and sets `errno` to `ERANGE`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

["strtol" on page 497](#)

["errno" on page 95](#)

["Overview of math.h" on page 197](#)

For example of `strtold()` usage refer to ["Example of `strtod\(\)` and `strtold\(\)` usage." on page 495](#).

strtoll

Character array conversion to integer value of type long long int.

```
#include <stdlib.h>
unsigned long int strtoul(const char *nptr,
                           char **endptr, int base);
```

nptr	const char *	A Null terminated array to convert
endptr	char **	A pointer to a position in nptry that is not convertible.
base	int	A numeric base between 2 and 36

Remarks

The `strtoll()` function converts a character array, pointed to by `nptr`, expected to represent an integer expressed in radix `base` to an integer value of type long long int. If the sequence pointed to by `nptr` is a minus sign, the value resulting from the conversion is negated in the return value.

The `base` argument in `strtoll()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then `strtoll()` converts the character array based on its format. Character arrays beginning with '0' are assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to a long int value.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`strtoll()` returns an unsigned integer value of type long long int. If the converted value is less than `LLONG_MIN`, `strtoll()` returns `LLONG_MIN` and sets `errno` to `ERANGE`. If the converted value is greater than `LLONG_MAX`, `strtoll()` returns `LLONG_MAX` and sets `errno` to `ERANGE`. The `LLONG_MIN` and `LLONG_MAX` macros are defined in `limits.h`

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

- [“strtod” on page 493](#)
- [“strtol” on page 497](#)
- [“errno” on page 95](#)
- [“Integral type limits” on page 189](#)
- [“Overview of math.h” on page 197](#)

For example of strtoll() usage refer to [“Example of strtol\(\), strtoul\(\), strtoll\(\), and strtoull\(\) usage” on page 499](#).

strtoul

Character array conversion to integer value of type unsigned long int.

```
#include <stdlib.h>
unsigned long int strtoul(const char *nptr,
                          char **endptr, int base);
```

nptr	const char *	A Null terminated array to convert
endptr	char **	A pointer to a position in nptry that is not convertible.
base	int	A numeric base between 2 and 36

Remarks

The `strtoul()` function converts a character array, pointed to by `nptr`, to an integer value of type `unsigned long int`, in `base`. If the sequence pointed to by `nptr` is a minus sign, the value resulting from the conversion is negated in the return value.

The `base` argument in `strtoul()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then `strtol()` and `strtoul()` convert

the character array based on its format. Character arrays beginning with '0' are assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to the functions' respective types.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

The function `strtoul()` returns an `unsigned integer` value of type `unsigned long int` (which may have a negative sign if the original string was negative.) If the converted value is greater than `ULONG_MAX`, `strtoul()` returns `ULONG_MAX` and sets `errno` to `ERANGE`. The `ULONG_MAX` macro is defined in `limits.h`

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

["strtod" on page 493](#)

["strtol" on page 497](#)

["errno" on page 95](#)

["Integral type limits" on page 189](#)

["Overview of math.h" on page 197](#)

For example of `strtoll()` usage refer to ["Example of strtol\(\), strtoul\(\), strtoll\(\), and strtoull\(\) usage" on page 499](#).

strtoull

Character array conversion to integer value of type `unsigned long long int`.

```
#include <stdlib.h>
unsigned long int strtoul(const char *nptr,
                          char **endptr, int base);
```

nptr	const char *	A Null terminated array to convert
endptr	char **	A pointer to a position in nptr that is not convertible.
base	int	A numeric base between 2 and 36

Remarks

The `strtoull()` function converts a character array, pointed to by `nptr`, expected to represent an integer expressed in radix base to an integer value of type `unsigned long long int`. If the sequence pointed to by `nptr` is a minus sign, the value resulting from the conversion is negated in the return value.

The base argument in `strtoull()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than base are permitted. If base is 0, then `strtoull()` converts the character array based on its format. Character arrays beginning with '0' are assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to a `long int` value.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

The function `strtoull()` returns an `unsigned integer` value of type `unsigned long long int` (which may have a negative sign if the original string was negative.) If the converted value is greater than `ULLONG_MAX`, `strtoull()` returns `ULLONG_MAX` and sets `errno` to `ERANGE`. The `ULLONG_MAX` macro is defined in `limits.h`

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

["strtoul" on page 502](#)

["strtoll" on page 501](#)

For example of strtoll() usage refer to “[Example of strtol\(\), strtoul\(\), strtoll\(\), and strtoull\(\) usage](#)” on page 499.

system

Environment list assignment.

```
#include <stdlib.h>  
  
int system(const char *string);
```

NOTE The system() function is an empty function on MacOS t

string	const char *	A OS system command
--------	--------------	---------------------

system() returns zero if successful or minus one on failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“getenv” on page 480](#)

vec_calloc

Clears and allocates memory on a 16 byte alignment.

```
#include <stdlib.h>  
  
void * vec_calloc(size_t nmemb, size_t size);
```

nmemb	size_t	Number of elements
size	size_t	The size of the elements

Remarks

The `vec_calloc()` function allocates contiguous space for `nmemb` elements of size. The space is initialized with zeroes.

`vec_calloc()` returns a pointer to the first byte of the memory area allocated. `vec_calloc()` returns a null pointer (`NULL`) if no space could be allocated.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“calloc” on page 473](#)

[“vec_free” on page 506](#)

[“vec_malloc” on page 507](#)

[“vec_realloc” on page 508](#)

vec_free

Frees memory allocated by `vec_malloc`, `vec_calloc` and `vec_realloc`

```
#include <stdlib.h>
void    vec_free(void *ptr);
```

ptr	void *	A pointer to the allocated memory
-----	--------	-----------------------------------

Remarks

The `vec_free()` function releases a previously allocated memory block, pointed to by `ptr`, to the heap. The `ptr` argument should hold an address returned by the memory allocation functions `vec_calloc()`, `vec_malloc()`, or `vec_realloc()`. Once the memory block `ptr` points to has been released, it is no longer valid. The `ptr` variable should not be used to reference memory again until it is assigned a value from the memory allocation functions.

There is no return.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“free” on page 479](#)
- [“vec_calloc” on page 505](#)
- [“vec_malloc” on page 507](#)
- [“vec_realloc” on page 508](#)

vec_malloc

Allocates memory on a 16 byte alignment.

```
#include <stdlib.h>
void * vec_malloc(size_t size);
```

size	size_t	The size in bytes of the allocation
------	--------	-------------------------------------

Remarks

The `vec_malloc()` function allocates a block of contiguous heap memory `size` bytes large.

`vec_malloc()` returns a pointer to the first byte of the allocated block if it is successful and return a null pointer if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“malloc” on page 484](#)
- [“vec_free” on page 506](#)
- [“vec_calloc” on page 505](#)
- [“vec_realloc” on page 508](#)

vec_realloc

Reallocates memory on a 16 byte alignment.

```
#include <stdlib.h>
void * vec_realloc(void * ptr, size_t size);
```

ptr	void *	A pointer to an allocated block of memory
size	size_t	The size of memory to reallocate

`vec_realloc()` returns a pointer to the new block if it is successful and `size` is greater than 0. `realloc()` returns a null pointer if it fails or `size` is 0.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“realloc” on page 491](#)
- [“vec_free” on page 506](#)
- [“vec_calloc” on page 505](#)
- [“vec_malloc” on page 507](#)

wcstombs

Translate a `wchar_t` type character array to a multibyte character array encoded as defined by the `LC_CTYPE` category of the current locale.

```
#include <stdlib.h>
size_t wcstombs(char *s, const
                 wchar_t *pwcs, size_t n);
```

s	char *	A multibyte string buffer
---	--------	---------------------------

pwcs	const wchar_t *	A pointer to a wide character string to be converted
n	size_t	The maximum length to convert

Remarks

The MSL implementation of the `wcstombs()` function converts a character array containing `wchar_t` type Unicode characters to a character array containing multibyte characters encoded as defined by the `LC_CTYPE` category of the current locale. The `wchar_t` type is defined in `stddef.h`. Each wide character is converted as if by a call to the `wctomb()` function. No more than `n` bytes will be modified in the array pointed to by `s`.

The function terminates prematurely if a `null` character is reached.

`wcstombs()` returns the number of bytes modified in the character array pointed to by `s`, not including a terminating null character, if any.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Locale specification” on page 191](#), [“mbstowcs” on page 486](#)

wctomb

Translate a `wchar_t` type to a multibyte character encoded as defined by the `LC_CTYPE` category of the current locale.

```
#include <stdlib.h>

int wctomb(char *s, wchar_t wchar);
```

s	char *	A multibyte string buffer
wchar	wchar_t	A wide character to convert

Remarks

The MSL implementation of the `wctomb()` function converts a `wchar_t` type Unicode character to a multibyte character encoded as defined by the

LC_CTYPE category of the current locale. If `s` is not a null pointer, the encoded multibyte character is stored in the array whose first element is pointed to by `s`. At most `MB_CUR_MAX` characters are stored. If `wchar` is a null wide character, a null byte is stored.

`wctomb()` returns `1` if `s` is not null and returns `0`, otherwise it returns the number of bytes that are contained in the multibyte character stored in the array whose first element is pointed to by `s`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Locale specification” on page 191](#)

[“mbtowc” on page 487](#)

Non Standard <stdlib.h> Functions

Various non standard functions are included in the header `stdlib.h` for legacy source code and compatibility with operating system frameworks and application programming interfaces.

- For the function `gcvt` see [“gcvt” on page 105](#), for a full description.
- For the function `itoa` see [“itoa” on page 107](#), for a full description.
- For the function `itow` see [“itow” on page 108](#), for a full description.
- For the function `ltoa` see [“ltoa” on page 109](#), for a full description.
- For the function `makepath` see [“makepath” on page 110](#), for a full description.
- For the function `splitpath` see [“splitpath” on page 113](#), for a full description.
- For the function `ultoa` see [“ultoa” on page 127](#), for a full description.
- For the function `wtoi` see [“wtoi” on page 136](#), for a full description.

string.h

The `string.h` header file provides functions for comparing, copying, concatenating, and searching character arrays and arrays of larger items.

Overview of `string.h`

The `string.h` header file provides multiple functions with and without length limits, with and without case sensitivity for comparing, copying, concatenating, and searching character arrays and generic arrays of larger items in memory.

The function naming convention used in `string.h` determines the type of data structure(s) a function manipulates.

A function with an `str` prefix operates on character arrays terminated with a null character ('`\0`'). The `str` functions are

- [“`strcat`” on page 518](#) concatenates strings
- [“`strchr`” on page 519](#) searches by character
- [“`strcmp`” on page 520](#) compares strings
- [“`strcpy`” on page 524](#) copies strings
- [“`strcoll`” on page 522](#) compares string lexicographically
- [“`strcspn`” on page 525](#) find a substring in a string
- [“`strerror`” on page 527](#) retrieves an error message from an `errno` variable
- [“`strerror_r`” on page 528](#) a reentrant version of `strerror`
- [“`strlen`” on page 528](#) returns string's length
- [“`strpbrk`” on page 534](#) look for an occurrence of a character from one string in another
- [“`strrchr`” on page 535](#) searches a string for a character
- [“`strspn`” on page 536](#) search for a character not in one string in another
- [“`strstr`” on page 537](#) searches a string for a string
- [“`strtok`” on page 539](#) retrieves the next token or substring

-
- [“strxfrm” on page 540](#) transform a string to a locale

A function with an `strn` prefix operates on character arrays of a length specified as a function argument. The `strn` functions are:

- [“strncat” on page 529](#), concatenates strings with length specified.
- [“strcmp” on page 531](#) string compare with length specified
- [“strncpy” on page 532](#) string copy with length specified

A function with a `mem` prefix operates on arrays of items or contiguous blocks of memory. The size of the array or block of memory is specified as a function argument. The `mem` functions are:

- [“memchr” on page 512](#) searches a memory block for a character
- [“memcmp” on page 515](#) compares a memory block
- [“memcpy” on page 516](#) copies a memory block
- [“memmove” on page 517](#) moves a memory block
- [“memset” on page 518](#) sets a value for a memory block

The nonstandard functions with a ‘stri’ prefix operate on strings ignoring case.

memchr

Search for an occurrence of a character.

```
#include <string.h>
void *memchr(const void *s, int c, size_t n);
```

s	const void *	The memory to search
c	int	The char to search for
n	size_t	The maximum length to search

Remarks

The `memchr()` function looks for the first occurrence of `c` in the first `n` characters of the memory area pointed to by `s`.

`memchr()` returns a pointer to the found character, or a null pointer (`NULL`) if `c` cannot be found.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strchr” on page 519](#)

[“ strrchr” on page 535](#)

string.h*Overview of string.h*

Listing 36.1 Example of memchr() usage.

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

#define ARRSIZE 100

int main(void)
{
    // s1 must by same length as s2 for this example!
    static char s1[ARRSIZE] = "laugh* giggle 231!";
    static char s2[ARRSIZE] = "grunt sigh# snort!";
    char dest[ARRSIZE];
    char *strptr;
    int len1, len2, lendift;

    // Clear destination string using memset()
    memset( (char *)dest, '\0', ARRSIZE);

    // String lengths are needed by the mem functions
    // Add 1 to include the terminating '\0' character
    len1 = strlen(s1) + 1;
    len2 = strlen(s2) + 1;
    lendift = strlen(dest) + 1;

    printf(" s1=%s\n s2=%s\n dest=%s\n\n", s1, s2, dest);

    if (memcmp( (char *)s1, (char *)s2, len1) > 0)
        memcpy( (char *)dest, (char *)s1, len1);
    else
        memcpy( (char *)dest, (char *)s2, len2);

    printf(" s1=%s\n s2=%s\n dest=%s\n\n", s1, s2, dest);

    // copy s1 onto itself using memchr() and memmove()
    strptr = (char *)memchr( (char *)s1, '*', len1);
    memmove( (char *)strptr, (char *)s1, len1);

    printf(" s1=%s\n s2=%s\n dest=%s\n\n", s1, s2, dest);

    return 0;
}
```

Output:

```
s1=laugh* giggle 231!
s2=grunt sigh# snort!
```

```
dest=

s1=laugh* giggle 231!
s2=grunt sigh# snort!
dest=laugh* giggle 231!

s1=laughlaugh* giggle 231!
s2=grunt sigh# snort!
dest=laugh* giggle 231!
```

memcmp

Compare two blocks of memory.

```
#include <string.h>

int memcmp(const void *s1,
           const void *s2, size_t n);
```

s1	const void *	The memory to compare
s2	const void *	The comparison memory
n	size_t	The maximum length to compare

Remarks

The `memcmp()` function compares the first `n` characters of `s1` to `s2` one character at a time.

`memcmp()` returns a zero if all `n` characters pointed to by `s1` and `s2` are equal.

`memcmp()` returns a negative value if the first non-matching character pointed to by `s1` is less than the character pointed to by `s2`.

`memcmp()` returns a positive value if the first non-matching character pointed to by `s1` is greater than the character pointed to by `s2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcmp” on page 520](#)

[“strncmp” on page 531](#)

For example of memcmp() usage refer to [“Example of memchr\(\) usage.” on page 514.](#)

memcpy

Copy a contiguous memory block.

```
#include <string.h>
void *memcpy(void *dest,
             const void *source, size_t n);
```

dest	void *	The destination memory
source	const void *	The source to copy
n	size_t	The maximum length to copy

Remarks

The `memcpy()` function copies the first `n` characters from the item pointed to by `source` to the item pointed to by `dest`. The behavior of `memcpy()` is undefined if the areas pointed to by `dest` and `source` overlap. The `memmove()` function reliably copies overlapping memory blocks.

`memcpy()` returns the value of `dest`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memmove” on page 517](#)

[“strcpy” on page 524](#)

[“strncpy” on page 532](#)

For example of `memcpy()` usage refer to [“Example of memchr\(\) usage.” on page 514.](#)

memmove

Copy an overlapping contiguous memory block.

```
#include <string.h>
void *memmove(void *dest, const void *source, size_t n);
```

dest	void *	The Memory destination
source	const void *	The source to be moved
n	size_t	The maximum length to move

Remarks

The `memmove()` function copies the first `n` characters of the item pointed to by `source` to the item pointed to by `dest`.

Unlike `memcpy()`, the `memmove()` function safely copies overlapping memory blocks.

`memmove()` returns the value of `dest`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memcpy” on page 516](#)

[“memset” on page 518](#)

[“strcpy” on page 524](#)

[“strncpy” on page 532](#)

For example of `memmove()` usage refer to [“Example of memchr\(\) usage.” on page 514](#).

string.h

Overview of string.h

memset

Set the contents of a block of memory to the value of a single character.

```
#include <string.h>
void *memset(void *dest, int c, size_t n);
```

dest	void *	The destination memory
c	int	The char to set
n	size_t	The maximum length to set

Remarks

The `memset()` function assigns `c` to the first `n` characters of the item pointed to by `dest`.

`memset()` returns the value of `dest`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

For example of `memset()` usage refer to [“Example of `memchr\(\)` usage.” on page 514](#).

strcat

Concatenate two character arrays.

```
#include <string.h>
char *strcat(char *dest, const char *source);
```

dest	char *	The destination string
source	const char *	The source to append

Remarks

The `strcat()` function appends a copy of the character array pointed to by `source` to the end of the character array pointed to by `dest`. The `dest` and

source arguments must both point to null terminated character arrays.
strcat() null terminates the resulting character array.

strcat() returns the value of dest.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcpy” on page 524](#)

Listing 36.2 Example of strcat() usage.

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

int main(void)
{
    char s1[100] = "The quick brown fox ";
    static char s2[] = "jumped over the lazy dog.';

    strcat(s1, s2);
    puts(s1);

    return 0;
}
```

Output:

The quick brown fox jumped over the lazy dog.

strchr

Search for an occurrence of a character.

```
#include <string.h>

char *strchr(const char *s, int c);
```

s	const char *	The string to search
c	int	The char to search for

string.h

Overview of string.h

Remarks

The `strchr()` function searches for the first occurrence of the character `c` in the character array pointed to by `s`. The `s` argument must point to a null terminated character array.

`strchr()` returns a pointer to the successfully located character. If it fails, `strchr()` returns a null pointer (`NULL`).

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memchr” on page 512](#)

[“strrchr” on page 535](#)

Listing 36.3 Example of `strchr()` usage.

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

int main(void)
{
    static char s[] = "tree * tomato eggplant garlic";
    char *strptr;

    strptr = strchr(s, '*');
    puts(strptr);

    return 0;
}
```

Output:

* tomato eggplant garlic

strcmp

Compare two character arrays.

```
#include <string.h>

int strcmp(const char *s1, const char *s2);
```

s1	const char *	The string to compare
s2	const char *	The comparison string

Remarks

The `strcmp()` function compares the character array pointed to by `s1` to the character array pointed to by `s2`. Both `s1` and `s2` must point to null terminated character arrays.

`strcmp()` returns a zero if `s1` and `s2` are equal, a negative value if `s1` is less than `s2`, and a positive value if `s1` is greater than `s2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memcmp” on page 515](#)

[“strcoll” on page 522](#)

[“strncmp” on page 531](#)

string.h

Overview of string.h

Listing 36.4 Example of strcmp() usage.

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

int main (void)
{
    static char s1[] = "butter", s2[] = "olive oil";
    char dest[20];

    if (strcmp(s1, s2) < 0)
        strcpy(dest, s2);
    else
        strcpy(dest, s1);

    printf(" s1=%s\n s2=%s\n dest=%s\n", s1, s2, dest);

    return 0;
}
```

Output:

```
s1=butter
s2=olive oil
dest=olive oil
```

strcoll

Compare two character arrays according to locale.

```
#include <string.h>

int strcoll(const char *s1, const char *s2);
```

s1	const char *	The string to compare
s2	const char *	The comparison string

Remarks

The `strcoll()` function compares two character arrays based on the `LC_COLLATE` component of the current locale.

The Metrowerks C implementation of `strcoll()` compares two character arrays using `strcmp()`. It is included in the string library to conform to the ANSI C Standard Library specification.

`strcoll()` returns zero if `s1` is equal to `s2`, a negative value if `s1` is less than `s2`, and a positive value if `s1` is greater than `s2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Locale specification” on page 191](#)

[“memcmp” on page 515](#)

[“strcmp” on page 520](#)

[“strncmp” on page 531](#),

Listing 36.5 Example of strcoll() usage.

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

int main(void)
{
    static char s1[] = "aardvark", s2[] = "xylophone";
    int result;

    result = strcoll(s1, s2);

    if (result < 1)
        printf("%s is less than %s\n", s1, s2);
    else
        printf("%s is equal or greater than %s\n", s1, s2);

    return 0;
}
```

Output:

aardvark is less than xylophone

strcpy

Copy one character array to another.

```
#include <string.h>

char *strcpy(char *dest, const char *source);
```

dest	char *	The destination string
source	const char *	The string being copied

Remarks

The `strcpy()` function copies the character array pointed to by `source` to the character array pointed to `dest`. The `source` argument must point to a null terminated character array. The resulting character array at `dest` is null terminated as well.

If the arrays pointed to by `dest` and `source` overlap, the operation of `strcpy()` is undefined.

`strcpy()` returns the value of `dest`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memcpy” on page 516](#)

[“memmove” on page 517](#)

[“strncpy” on page 532](#)

Listing 36.6 Example of strcpy() usage.

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

int main(void)
{
    char d[30] = "";
    static char s[] = "Metrowerks";

    printf(" s=%s\n d=%s\n", s, d);
    strcpy(d, s);
    printf(" s=%s\n d=%s\n", s, d);

    return 0;
}
```

Output:

```
s=Metrowerks
d=
s=Metrowerks
d=Metrowerks
```

strcspn

Find the first character in one string that is in another.

```
#include <string.h>
size_t strcspn(const char *s1, const char *s2);
```

s1	const char *	The string to count
s2	const char *	The list string of character to search for

Remarks

The `strcspn()` function finds the first character in the null terminated character string `s1` that is also in the null terminated string `s2`. For this purpose, the null terminators are considered part of the strings. The function starts examining characters at the beginning of `s1` and continues searching until a character in `s1` matches a character in `s2`.

`strcspn()` returns the index of the first character in `s1` that matches a character in `s2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“`strpbrk`” on page 534](#)

[“`strspn`” on page 536](#)

Listing 36.7 Example of `strcspn()` usage.

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

int main(void)
{
    static char s1[] = "chocolate *cinnamon* 2 ginger";
    static char s2[] = "1234*";
    int i;

    printf(" s1 = %s\n s2 = %s\n", s1, s2);
    i = strcspn(s1, s2);
    printf("Index returned by strcspn %d\n", i);
    printf("Indexed character = %c\n", s1[i]);

    return 0;
}
```

Output:

```
s1 = chocolate *cinnamon* 2 ginger
s2 = 1234*
Index returned by strcspn 10
Indexed character = *
```

strerror

Translate an error number into an error message.

```
#include <string.h>
char *strerror(int errnum);
```

errnum	int	The error number to be translated.
--------	-----	------------------------------------

Remarks

The `strerror()` function returns a pointer to a null terminated character array that contains an error message. The array pointed to may not be modified by the program, but may be overwritten by a subsequent call to the `strerror` function.

`strerror()` returns a pointer to a null terminated character array containing an error message that corresponds to `errnum`. Although normally the integer value in `errnum` will come from the global variable `errno`, `strerror()` will provide a message translation for any value of type `int`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 36.8 Example of strerror() usage.

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

int main(void)
{
    puts(strerror(8));
    puts(strerror(ESIGPARM));

    return 0;
}
```

Output:
unknown error (8)
Signal Error

strerror_r

Translate an error number into an error message.

```
#include <string.h>
int strerror_r(int errnum, char *str, size_t bufflen);
```

errnum	int	The error number to be translated.
str	char *	
bufflen	size_t	

Remarks

The `strerror_r()` function provides the same service as “[strerror](#)” on page [527](#) but is reentrant. The difference is that `strerror()` would return a pointer to the error string, and that pointer was internal to the library implementation. For `strerror_r()`, the caller provides the storage `str` and the size of the storage `bufflen`.

On a successful call to `strerror_r()`, the function result is zero. If any error occurs, the function result is an error code.

This function may require extra library support.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strerror” on page 527](#)

strlen

Compute the length of a character array.

```
#include <string.h>
size_t strlen(const char *s);
```

s1	const char *	The string to evaluate
----	--------------	------------------------

Remark

The `strlen()` function computes the number of characters in a null terminated character array pointed to by `s`. The null character (' \0 ') is not added to the character count.

`strlen()` returns the number of characters in a character array not including the terminating null character.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 36.9 Example of `strlen()` usage.

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

int main(void)
{
    static char s[] = "antidisestablishmentarianism";

    printf("The length of %s is %ld.\n", s, strlen(s));

    return 0;
}
```

Output:

The length of antidisestablishmentarianism is 28.

strncat

Append a specified number of characters to a character array.

```
#include <string.h>

char *strncat(char *dest, const char *source, size_t n);
```

dest	char *	The destination string
source	const char *	The source to append
n	size_t	The maximum length to append

Remarks

The `strncat()` function appends a maximum of `n` characters from the character array pointed to by `source` to the character array pointed to by `dest`. The `dest` argument must point to a null terminated character array. The `source` argument does not necessarily have to point to a null terminated character array.

If a null character is reached in `source` before `n` characters have been appended, `strncat()` stops.

When done, `strncat()` terminates `dest` with a null character (' \0 ').

`strncat()` returns the value of `dest`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“strcat” on page 518](#)

Listing 36.10 Example of strncat() usage.

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

int main(void)
{
    static char s1[100] = "abcdefghijklmnoprstuv";
    static char s2[] = "wxyz0123456789";

    strncat(s1, s2, 4);
    puts(s1);

    return 0;
}
```

Output:

abcdefghijklmnoprstuvwxyz

strcmp

Compare a specified number of characters.

```
#include <string.h>  
  
int strcmp(const char *s1, const char *s2, size_t n);
```

s1	const char *	The string to compare
s2	const char *	The comparison string
n	size_t	The maximum number of characters to compare

Remarks

The `strcmp()` function compares `n` characters of the character array pointed to by `s1` to `n` characters of the character array pointed to by `s2`. Neither `s1` nor `s2` needs to be null terminated character arrays.

The function stops prematurely if it reaches a null character before `n` characters have been compared.

`strcmp()` returns a zero if the first `n` characters of `s1` and `s2` are equal, a negative value if `s1` is less than `s2`, and a positive value if `s1` is greater than `s2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memcmp” on page 515](#)

[“strcmp” on page 520](#)

string.h

Overview of string.h

Listing 36.11 Example of strncmp() usage.

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

int main(void)
{
    static char s1[] = "12345anchor", s2[] = "12345zebra";

    if (strncmp(s1, s2, 5) == 0)
        printf("%s is equal to %s\n", s1, s2);
    else
        printf("%s is not equal to %s\n", s1, s2);

    return 0;
}
```

Output:

```
12345anchor is equal to 12345zebra
```

strncpy

Copy a specified number of characters.

```
#include <string.h>
char *strncpy(char *dest, const char *source, size_t n);
```

dest	char *	The destination string
source	const char *	The source to copy
n	size_t	The maximum length to copy

Remarks

The `strncpy()` function copies a maximum of `n` characters from the character array pointed to by `source` to the character array pointed to by `dest`. Neither `dest` nor `source` need necessarily point to null terminated character arrays. Also, `dest` and `source` must not overlap.

If a null character ('\0') is reached in `source` before `n` characters have been copied, `strncpy()` continues padding `dest` with null characters until `n` characters have been added to `dest`.

The function does not terminate `dest` with a null character if `n` characters are copied from `source` before reaching a null character.

`strncpy()` returns the value of `dest`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memcpy” on page 516](#)

[“memmove” on page 517](#)

[“strcpy” on page 524](#)

Listing 36.12 Example of strncpy usage.

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

int main(void)
{
    char d[50];
    static char s[] = "123456789ABCDEFG";

    strncpy(d, s, 9);
    puts(d);

    return 0;
}
```

Output:
123456789

strpbrk

Look for the first occurrence of any one of an array of characters in another.

```
#include <string.h>
char *strpbrk(const char *s1, const char *s2);
```

s1	const char *	The string being searched
s2	const char *	A list of characters to search for

Remarks

The `strpbrk()` function searches the character array pointed to by `s1` for the first occurrence of a character in the character array pointed to by `s2`.

Both `s1` and `s2` must point to null terminated character arrays.

`strpbrk()` returns a pointer to the first character in `s1` that matches any character in `s2`, and returns a null pointer (`NULL`) if no match was found.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcspn” on page 525](#)

Listing 36.13 Example of strpbrk usage.

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

int main(void)
{
    static char s1[] = "orange banana pineapple *plum";

    static char s2[] = "*%#$";
    puts(strpbrk(s1, s2));

    return 0;
}
```

Output:

```
*plum
```

strrchr

Searches a string for the last occurrence of a character.

```
#include <string.h>

char *strrchr(const char *s, int c);
```

s	const char *	The string to search
c	int	A character to search for

Remarks

The `strrchr()` function searches for the last occurrence of `c` in the character array pointed to by `s`. The `s` argument must point to a null terminated character array.

`strrchr()` returns a pointer to the character found or returns a null pointer (`NULL`) if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memchr” on page 512](#)

[“strchr” on page 519](#)

Listing 36.14 Example of `strrchr()` usage.

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

int main(void)
{
    static char s[] = "Marvin Melany Metrowerks";
    puts(strrchr(s, 'M'));

    return 0;
}
```

Output:Metrowerks

strspn

Find the first character in one string that is not in another.

```
#include <string.h>

size_t strspn(const char *s1, const char *s2);
```

s1	const char *	The string to search
s2	const char *	A list of characters to look for

Remarks

The `strspn()` function finds the first character in the null terminated character string `s1` that is not in the null terminated string `s2`. The function starts examining characters at the beginning of `s1` and continues searching until a character in `s1` does not match any character in `s2`.

Both `s1` and `s2` must point to null terminated character arrays.

`strspn()` returns the index of the first character in `s1` that does not match a character in `s2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“`strpbrk`” on page 534](#)

[“`strcspn`” on page 525](#)

Listing 36.15 Example of `strspn()` usage.

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

int main(void)
{
    static char s1[] = "create *build* construct";
    static char s2[] = "create *";

    printf(" s1 = %s\n s2 = %s\n", s1, s2);
    printf(" %d\n", strspn(s1, s2));

    return 0;
}
```

Output:

```
s1 = create *build* construct
s2 = create *
8
```

strstr

Search for a character array within another.

```
#include <string.h>

char *strstr(const char *s1, const char *s2);
```

s1	const char *	The string to search
s2	const char *	The string to search for

Remarks

The `strstr()` function searches the character array pointed to by `s1` for the first occurrence of the character array pointed to by `s2`.

Both `s1` and `s2` must point to null terminated (' \0 ') character arrays.

`strstr()` returns a pointer to the first occurrence of `s2` in `s1` and returns a null pointer (NULL) if `s2` cannot be found.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memchr” on page 512](#)

[“strchr” on page 519](#)

Listing 36.16 Example of strstr() usage.

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

int main(void)
{
    static char s1[] = "tomato carrot onion";
    static char s2[] = "on";
    puts(strstr(s1, s2));

    return 0;
}
```

Output:
onion

strtok

Extract tokens within a character array.

```
#include <string.h>  
  
char *strtok(char *str, const char *sep);
```

str	char *	The string to be separate
sep	const char *	The separator string

Remarks

The `strtok()` function divides a null terminated character array pointed to by `str` into separate “tokens”. The `sep` argument points to a null terminated character array containing one or more separator characters. The tokens in `str` are extracted by successive calls to `strtok()`.

`Strtok()` works by a sequence of calls to the `strtok()` function. The first call is made with the string to be divided into tokens, as the first argument. Subsequent calls use `NULL` as the first argument and returns pointers to successive tokens of the separated string.

The first call to `strtok()` causes it to search for the first character in `str` that does not occur in `sep`. If no character other than those in the `sep` string can be found, `strtok()` returns a null pointer (`NULL`). If no characters from the `sep` string are found it returns a pointer to the original string. Otherwise the function returns a pointer to the beginning of this first token.

Subsequent calls to `strtok()` are made with a `NULL` `str` argument causing it to return pointers to successive tokens in the original `str` character array. If no further tokens exist, `strtok()` returns a null pointer.

Both `str` and `sep` must be null terminated character arrays.

The `sep` argument can be different for each call to `strtok()`. `strtok()` modifies the character array pointed to by `str`.

When first called `strtok()` returns a pointer to the first token in `str`. If nothing but separator characters are found `strtok` returns a null pointer.

Subsequent calls to `strtok()` with a `NULL` `str` argument causes `strtok()` to return a pointer to the next token or return a null pointer (`NULL`) when no more tokens exist.

string.h

Overview of string.h

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

Listing 36.17 Example of strtok() usage.

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

int main(void)
{
    static char s[50] = "(ape+bear)*(cat+dog)";
    char *token;
    char *separator = "()+*";

    /* first call to strtok() */
    token = strtok(s, separator);

    while(token != NULL)
    {
        puts(token);
        token = strtok(NULL, separator);
    }

    return 0;
}
```

Output:

```
ape
bear
cat
dog
```

strxfrm

Transform a locale-specific character array.

```
#include <string.h>
size_t strxfrm(char *dest, const char *source, size_t n);
```

dest	char *	The destination string
source	const char *	The source to be transformed
n	size_t	The maximum length to transform

Remarks

The `strxfrm()` function copies characters from the character array pointed to by `source` to the character array pointed to by `dest`, transforming each character as specified by the `LC_COLLATE` component of the current locale. The `strxfrm` function transforms the string pointed to by `source` and places the resulting string into the array pointed to by `dest`. The transformation is such that if the `strcmp` function is applied to two transformed strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the `strcoll` function applied to the same two original strings.

The Metrowerks C implementation of `strxfrm()` copies a maximum of `n` characters from the character array pointed to by `source` to the character array pointed to by `dest` using the `strncpy()` function. It is included in the string library to conform to the ANSI C Standard Library specification.

`strxfrm()` returns the length of `dest` after it has received `source`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Locale specification” on page 191](#)

[“strcpy” on page 524](#)

Listing 36.18 Example of strxfrm() usage.

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

int main(void)
{
    char d[50];
    static char s[] = "123456789ABCDEFG";
    size_t result;

    result = strxfrm(d, s, 30);

    printf("%d characters copied: %s\n", result, d);

    return 0;
}
```

Output:

```
16 characters copied: 123456789ABCDEFG
```

Non Standard <string.h> Functions

Various non standard functions are included in the header `string.h` for legacy source code and compatibility with operating system frameworks and application programming interfaces.

- For the function `strdup` see [“`strdup`” on page 116](#) for a full description.
- For the function `stricmp` see [“`stricmp`” on page 116](#) for a full description.
- For the function `strlwr` see [“`strlwr`” on page 118](#) for a full description.
- For the function `strnicmp` see [“`strnicmp`” on page 120](#) for a full description.
- For the function `strnset` see [“`strnset`” on page 122](#) for a full description.
- For the function `strrev` see [“`strrev`” on page 123](#) for a full description.
- For the function `strset` see [“`strset`” on page 123](#) for a full description.
- For the function `strupr` see [“`strupr`” on page 125](#) for a full description.
- For the function `wcsdup` see [“`wcsdup`” on page 128](#), for a full description.
- For the function `wcsicmp` see [“`wcsicmp`” on page 129](#), for a full description.
- For the function `wcslwr` see [“`wcslwr`” on page 130](#), for a full description.
- For the function `wcsncmp` see [“`wcsncmp`” on page 132](#), for a full description.

- For the function `wcsnset` see “[wcsnset” on page 133](#), for a full description.
- For the function `wcsrev` see “[wcsrev” on page 134](#), for a full description.
- For the function `wcsset` see “[wcsset” on page 134](#), for a full description.
- For the function `wcspnp` see “[wcspnp” on page 135](#), for a full description.
- For the function `wcsupr` see “[wcsupr” on page 135](#), for a full description.
- For the function `wtoi` see “[wtoi” on page 136](#), for a full description.

string.h

Non Standard <string.h> Functions

tgmath.h

The header `tgmath.h` includes the header `math.h` and defines type-generic macros for those math functions.

Overview of tgmath.h

The `tgmath.h` header file consists of type-generic macros for most math functions listed in [“Type-Generic Macro for math functions” on page 545](#).

NOTE	Metrowerks Standard Library does not include complex types and complex type-generic equivalents
-------------	---

The macros in this header may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 37.1 Type-Generic Macro for math functions

Function	Type-Generic Macro	Function	Type-Generic Macro
acos	acos	acosh	acosh
asin	asin	asinh	asinh
atan	atan	atan2	atan2
atanh	atanh	cbrt	cbrt
ceil	ceil	copysign	copysign
cos	cos	cosh	cosh
erf	erf	erfc	erfc
exp	exp	exp2	exp2
expm1	expm1	fabs	fabs
fdim	fdim	floor	floor

Listing 37.1 Type-Generic Macro for math functions

Function	Type-Generic Macro	Function	Type-Generic Macro
fma	fma	fmax	fmax
fmin	fmin	fmod	fmod
frexp	frexp	hypot	hypot
ilogb	ilogb	ldexp	ldexp
lgamma	lgamma	llrint	llrint
llround	llround	log	log
log10	log10	log1p	log1p
log2	log2	logb	logb
lrint	lrint	lround	lround
nearbyint	nearbyint	nextafter	nextafter
nexttoward	nexttoward	pow	pow
remainder	remainder	remquo	remquo
rint	rint	round	round
scalbln	scalbln	scalbn	scalbn
sin	sin	sinh	sinh
sqrt	sqrt	tan	tan
tanh	tanh	tgamma	tgamma
trunc	trunc		

time.h

The `time.h` header file provides access to the computer system clock, date and time conversion functions, and time formatting functions.

Overview of `time.h`

The `time.h` facilities include:

- [“`struct tm`” on page 549](#) is a structure for storing time data.
- [“`tzname`” on page 550](#) an array that stores the time zone abbreviations
- [“`asctime`” on page 550](#) to convert a `tm` structure type to a char array
- [“`asctime_r`” on page 552](#) a reentrant version of `asctime`
- [“`clock`” on page 552](#) to determine the relative time since the system was started
- [“`ctime`” on page 555](#) to convert a `time_t` type to a char array
- [“`ctime_r`” on page 556](#) a reentrant version of `ctime`
- [“`difftime`” on page 556](#) to determine the difference between two times
- [“`gmtime`” on page 558](#) to determine Greenwich Mean Time
- [“`gmtime_r`” on page 559](#) a reentrant version of `gmtime`
- [“`localtime`” on page 560](#) to determine the local time
- [“`localtime_r`” on page 561](#) a reentrant version of `localtime`
- [“`mktime`” on page 561](#) to convert a `tm` structure to `time_t` type
- [“`strftime`” on page 562](#) to format time as a C string
- [“`time`” on page 568](#) to determine a number of seconds from a set time
- [“`tzset`” on page 569](#) internalizes the time zone to that of the application

Date and time

The `time.h` header file provides access to the computer system clock, date and time conversion functions, and formatting functions.

Three data types are defined in `time.h`: `clock_t`, `time_t`, and `tm`.

Type `clock_t`

The `clock_t` type is a numeric, system dependent type returned by the `clock()` function.

Type `time_t`

The `time_t` type is a system dependent type used to represent a calendar date and time.

Remarks

The type `time_t`'s range and precision are defined in the C standard as implementation defined. The MSL implementation uses an unsigned long for `time_t` and it represents the number of UTC seconds since 1900 January 1. If his value exceeds the size of the maximum value for unsigned long (`ULONG_MAX` = 4,294,967,295) the result is undefined. A value of greater than 136 for `tm_year` will exceed this limit. Similarly, since `time_t` is unsigned, negative values for `tm_year` are also out of range.

The ANSI/ISO C Standard does not specify a start date, therefore an arbitrarily chosen Jan. 1, 1970 is used for the MSL C Library. These routines are not meant to be intermixed with any specific API time functions. However some conversion constants are available in the OS specific headers (e.g. `time.mac.h`).

struct tm

The `struct tm` type contains a field for each part of a calendar date and time.

```
#include <time.h>

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;
};
```

Remarks

The `tm` structure members are listed [“Tm Structure Members.” on page 549](#).

The `tm_isdst` flag is positive if Daylight Savings Time is in effect, zero if it is not, and negative if such information is not available.

This structure may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 38.1 Tm Structure Members.

Field	Description	Range min - max
int tm_sec	Seconds after the minute	0 - 59
int tm_min	Minutes after the hour	0 - 59
int tm_hour	Hours after midnight	0 - 23
int tm_mday	Day of the month	1 - 31

time.h*Date and time*

This structure may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 38.1 Tm Structure Members.

int tm_mon	Months after January	0 - 11
int tm_year	Up to 136 years after 1900	1900 - 2036
int tm_wday	Days after Sunday	0 - 6
int tm_yday	Days after January 1	0 - 365
int tm_isdst	Daylight Savings Time flag	

tzname

The _tzname_ array contains the names (abbreviations) of the time zones for local standard time and DST.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

```
#include <time.h>
extern char *tzname[2];
```

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“tzset” on page 569](#)

asctime

Convert a tm structure to a character array.

```
#include <time.h>
char *asctime(const struct tm *timeptr);
```

timeptr	const struct tm *	A pointer to a tm structure that holds the time information
---------	-------------------	---

Remarks

The `asctime()` function converts a `tm` structure, pointed to by `timeptr`, to a character array. The `asctime()` and `ctime()` functions use the same calendar time format. This format, expressed as a `strftime()` format string is "%a %b %e %H:%M: %S %Y". For example: Tue Apr 4 15:17:23 2000.

`asctime()` returns a null terminated character array pointer containing the converted `tm` structure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ctime” on page 555](#)

[“strftime” on page 562](#)

Listing 38.2 Example of `asctime()` usage.

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

int main(void)
{
    time_t systime;
    struct tm *currtime;

    systime = time(NULL);
    currtime = localtime(&systime);

    puts(asctime(currtime));

    return 0;
}
```

Output:

Tue Nov 30 12:56:05 1993

asctime_r

Reentrant function to convert a `tm` structure to a character array.

```
#include <time.h>
char * asctime_r(const struct tm * tm, char * s);
```

<code>tm</code>	<code>const struct tm *</code>	
<code>s</code>	<code>char *</code>	

Remarks

The `asctime_r()` function provides a reentrant version of [“asctime” on page 550](#). The difference is that `asctime()` will return a pointer to the time string, and that pointer is internal to the library implementation. For `asctime_r()`, the caller provides the storage for string `s` and the size of the storage must be at least 26 characters long.

The `asctime_r()` function always returns the value `s`.

This function may require extra library support.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“asctime” on page 550](#)

[“ctime” on page 555](#)

[“strftime” on page 562](#)

clock

A program relative invocation of the system time.

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

Remarks

This function is used to obtain values of type `clock_t` which may be used to calculate elapsed times during the execution of a program. To compute the elapsed time in seconds, divide the `clock_t` value by `CLOCKS_PER_SEC`, a macro defined in `time.h`.

The programmer should be aware that `clock_t`, defined in `time.h`, has a finite size that varies depending upon the target system.

The `clock()` function returns a `clock_t` type value representing the approximation of time since the system was started. There is no error value returned if an error occurs.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 38.3 Example of clock() usage.

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

int main()
{
    clock_t start, end;
    double secs = 0;
    int stop = 0;

    fprintf( stderr, "Press enter to start");
    getchar();
    start = clock();

    while(stop != 'x')
    {
        fprintf( stderr, "Press enter to terminate");
        getchar();
        end = clock();
        secs = (double)(end - start) /CLOCKS_PER_SEC;
        fprintf( stderr, "Elapsed seconds = %f \n", secs);
        fprintf( stderr,"Press enter to start again ");
        fprintf(stderr, "or enter x to terminate:  ");
        stop = getchar();
        start = clock();
    }

    printf("\n** Program has terminated ** \n");
    return 0;
}
```

Output:

```
Press enter to start <enter>
Press enter to terminate <enter>
Elapsed seconds = 1.200000
Press enter to start again or enter x to terminate: <enter>
Press enter to terminate <enter>
Elapsed seconds = 1.033333
Press enter to start again or enter x to terminate: <x enter>

** Program has terminated **
```

ctime

Convert a `time_t` type to a character array.

```
#include <time.h>
char *ctime(const time_t *timer);
```

timer	const time_t *	The address of the <code>time_t</code> variable
-------	----------------	---

Remarks

The `ctime()` function converts a `time_t` type to a character array with the same format as generated by `asctime()`.

`ctime()` returns a null terminated character array pointer containing the converted `time_t` value.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“asctime” on page 550](#)

[“strftime” on page 562](#)

Listing 38.4 Example of `ctime()` usage.

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

int main(void)
{
    time_t systime;

    systime = time(NULL);
    puts(ctime(&systime));

    return 0;
}
```

Output:
Wed Jul 20 13:32:17 1994

ctime_r

Convert a `time_t` type to a character array but reentrant safe.

```
#include <time.h>
char * ctime_r(const time_t * timer, char * s);
```

timer	const time_t *	The address of the <code>time_t</code> variable
s	char *	The storage string

Remarks

The `ctime_r()` function provides the same service as [“ctime” on page 555](#). The difference is that `ctime()` would return a pointer to the time string, and that pointer was internal to the library implementation. For `ctime_r()`, the caller provides the storage for string `s` and the size of the storage must be at least 26 characters long.

The `ctime_r()` function always returns the value `s`.

This function may require extra library support.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“asctime” on page 550](#)

[“ctime” on page 555](#)

[“strftime” on page 562](#)

difftime

Compute the difference between two `time_t` types.

```
#include <time.h>
double difftime(time_t t1, time_t t2);
```

t1	time_t	A time_t variable to compare
t2	time_t	A time_t variable to compare

`difftime()` returns the difference of t1 minus t2 expressed in seconds.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 38.5 Example of difftime usage.

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

int main(void)
{
    time_t t1, t2;
    struct tm *currtme;
    double midnight;

    time(&t1);
    currtme = localtime(&t1);

    currtme->tm_sec = 0;
    currtme->tm_min = 0;
    currtme->tm_hour = 0;
    currtme->tm_mday++;

    t2 = mktime(currtme);

    midnight = difftime(t1, t2);
    printf("There are %f seconds until midnight.\n",
           midnight);

    return 0;
}
```

Output:

There are 27892.000000 seconds until midnight.

gmtime

Convert a `time_t` value to Coordinated Universal Time (UTC), which is the new name for Greenwich Mean Time.

```
#include <time.h>
struct tm *gmtime(const time_t *timer);
```

timer	const time_t *	The address of the <code>time_t</code> variable
-------	----------------	---

Remarks

The `gmtime` function converts the calendar time pointed to by `timer` into a broken-down time, expressed as UTC.

The `gmtime()` function returns a pointer to that object.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“gmtime_r” on page 559](#)

[“localtime” on page 560](#)

Listing 38.6 Example of gmtime usage.

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

int main(void)
{
    time_t systime;
    struct tm *utc;

    systime = time(NULL);
    utc = gmtime(&systime);

    printf("Universal Coordinated Time:\n");
    puts(asctime(utc));

    return 0;
}
```

Output:

```
Universal Coordinated Time:
Thu Feb 24 18:06:10 1994
```

gmtime_r

Convert a `time_t` value to Coordinated Universal Time (UTC), which is the new name for Greenwich Mean Time but is reentrant safe.

```
#include <time.h>

struct tm * gmtime_r(const time_t * timer, struct tm * tm);
```

timer	const time_t *	The address of the <code>time_t</code> variable
tm	struct tm *	A storage location for the converted time

Remarks

The `gmtime_r()` function provides the same service as “[gmtime on page 558](#)”. The difference is that `gmtime()` would return a pointer to the converted time, and that pointer was internal to the library implementation. For `gmtime_r()`, the caller provides the storage for the `tm` struct.

The `gmtime_r()` function always returns the value `tm`.

This function may require extra library support.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“gmtime” on page 558](#)

localtime

Convert a `time_t` type to a `struct tm` type.

```
#include <time.h>
struct tm *localtime(const time_t *timer);
```

timer	const time_t *	The address of the <code>time_t</code> variable
-------	----------------	---

Remarks

The `localtime()` function converts a `time_t` type, pointed to by `timer`, and returns it as a pointer to an internal `struct tm` type. The `struct tm` pointer is static; it is overwritten each time `localtime()` is called.

`localtime()` converts `timer` and returns a pointer to a `struct tm`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“mktime” on page 561](#)

Refer to the example for [“Example of difftime usage.” on page 557](#) for usage.

localtime_r

Convert a `time_t` type to a `struct tm` type but is reentrant.

```
#include <time.h>

struct tm * localtime_r(const time_t * timer, struct tm * tm);
```

timer	const <code>time_t</code> *	The address of the <code>time_t</code> variable
tn	<code>struct tm</code> *	The storage location

Remarks

The `localtime_r()` function provides the same service as [“localtime” on page 560](#) but is reentrant. The difference is that `localtime()` would return a pointer to the converted time, and that pointer was internal to the library implementation. For `localtime_r()`, the caller provides the storage for the `tm` struct.

The `localtime_r()` function always returns the value `tm`.

This function may require extra library support.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“localtime” on page 560](#)

[“mktime” on page 561](#)

mktime

Convert a `struct tm` item to a `time_t` type.

```
#include <time.h>

time_t mktime(struct tm *timeptr);
```

timeptr	<code>struct tm</code> *	The address of the <code>tm</code> structure
---------	--------------------------	--

Remarks

The `mktim()` function converts a `struct tm` type and returns it as a `time_t` type.

The function also adjusts the fields in `timeptr` if necessary. The `tm_sec`, `tm_min`, `tm_hour`, and `tm_day` are processed such that if they are greater than their maximum, the appropriate carry-overs are computed. For example, if `timeptr->tm_min` is 65, `timeptr->tm_hour` will be incremented by 1 and `timeptr->min` will be set to 5.

The function also computes the correct values for `timeptr->tm_wday` and `timeptr->tm_yday`.

`mktim()` returns the converted `tm` structure as a `time_t` type.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“localtime” on page 560](#)

For example of usage refer to the example for [“Example of difftime usage.” on page 557](#).

strftime

Format a `tm` structure.

```
#include <time.h>

size_t strftime(char *s, size_t maxsize,
                const char *format,
                const struct tm *timeptr);
```

s	char *	A string to hold the formatted time
maxsize	size_t	Max length of formatted string
format	const char *	The format string
timeptr	const struct tm*	The address of the time structure

Remarks

The `strftime()` function converts a `tm` structure to a character array using a programmer supplied format.

The `s` argument is a pointer to the array to hold the formatted time.

The `maxsize` argument specifies the maximum length of the formatted character array.

The `timeptr` argument points to a `tm` structure containing the calendar time to convert and format.

The `format` argument points to a character array containing normal text and format specifications similar to a `printf()` function format string. Format specifiers are prefixed with a percent sign (%). Doubling the percent sign (%%) will output a single %.

If any of the specified values are outside the normal range, the characters stored are unspecified.

In the “C” locale, the E and O modifiers are ignored. Also, some of the formats are dependent on the LC_TIME component of the current locale.

Refer to [“Strftime\(\) conversion characters” on page 563](#) for a list of format specifiers.

Listing 38.7 Strftime() conversion characters

Char	Description
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	The locale's appropriate date and time representation equivalent to the format string of "%A %B %d %T %Y".
C	The year divided by 100 and truncated to an integer, as a decimal number [00 - 99]
d	Day of the month as a decimal number [01 - 31].
D	The month date year, equivalent to "%m/%d/%y"
e	The day of the month as a decimal number; a single digit is preceded by a space.

Listing 38.7 strftime() conversion characters

Char	Description
F	The year, month and day separated by hyphens, the equivalent to "%Y-%m-%d"
g	The last 2 digits of the week-based year as a decimal number. For example: 03 99
G	The week-based year as a decimal number
h	The month name, equivalent to "%b"
H	The hour (24-hour clock) as a decimal number from 00 to 23.
I	The hour (12-hour clock) as a decimal number from 01 to 12
j	The day of the year as a decimal number from 001 to 366
m	The month as a decimal number from 01 to 12.
M	The minute as a decimal number from 00 to 59.
n	A newline character
p	Locale's equivalent of "am" or "pm".
r	The locale's 12-hour clock time, equivalent of "%I:%M:%S %p"
R	The hour, minute, equivalent to "%H:%M
S	The second as a decimal number from 00 to 59.
t	A horizontal-tab character
T	The hour minute second, equivalent to "%H:%M:%S"
u	The weekday as a decimal number 1 to 7, where Monday is 1.
U	The week number of the year as a decimal number from 00 to 53. Sunday is considered the first day of the week.
w	The weekday as a decimal number from 0 to 6. Sunday is (0) zero.
W	The week of the year as a decimal number from 00 to 51. Monday is the first day of the week.
x	The date representation of the current locale, equivalent to "%A %B %d %Y"
X	The time representation of the current locale, equivalent to "%T"

Listing 38.7 `Strftime()` conversion characters

Char	Description
y	The last two digits of the year as a decimal number.
Y	The year as a four digit decimal number.
z	The time zone offset from UTC. for example, -0430 is 4 hours 30 minutes behind UTC. Or nothing if the time zone is unknown.
Z	The locale's time zone name or abbreviation, or by no characters if no time zone is unknown.
%	The percent sign is displayed.

The `strftime()` function returns the total number of characters in the argument 's' if the total number of characters including the null character in the string argument 's' is less than the value of 'maxlen' argument. If it is greater, `strftime()` returns 0.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 38.8 Example of strftime() usage.

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

int main(void)
{
    time_t lclTime;
    struct tm *now;
    char ts[256]; /* time string */

    lclTime = time(NULL);
    now = localtime(&lclTime);

    strftime(ts, 256,
             "Today's abr.name is %a", now);
    puts(ts);

    strftime(ts, 256,
             "Today's full name is %A", now);
    puts(ts);

    strftime(ts, 256,
             "Today's aabr.month name is %b", now);
    puts(ts);

    strftime(ts, 256,
             "Today's full month name is %B", now);
    puts(ts);

    strftime(ts, 256,
             "Today's date and time is %c", now);
    puts(ts);
    strftime(ts, 256,
             "The day of the month is %d", now);
    puts(ts);

    strftime(ts, 256,
             "The 24-hour clock hour is %H", now);
    puts(ts);

    strftime(ts, 256,
             "The 12-hour clock hour is %H", now);
    puts(ts);
```

```
strftime(ts, 256,
"Today's day number is %j", now);
puts(ts);

strftime(ts, 256,
"Today's month number is %m", now);
puts(ts);

strftime(ts, 256,
"The minute is %M", now);
puts(ts);

strftime(ts, 256,
"The AM/PM is %p", now);
puts(ts);

strftime(ts, 256,
"The second is %S", now);
puts(ts);

strftime(ts, 256,
"The week number of the year,\n"
"starting on a Sunday is %U", now);
puts(ts);

strftime(ts, 256,
"The number of the week is %w", now);
puts(ts);

strftime(ts, 256, "The week number of the year,\n"
"starting on a Monday is %W", now);
puts(ts);

strftime(ts, 256, "The date is %x", now);
puts(ts);

strftime(ts, 256, "The time is %X", now);
puts(ts);

strftime(ts, 256,
"The last two digits of the year are %y", now);
puts(ts);

strftime(ts, 256, "The year is %Y", now);
puts(ts);
```

time.h

Date and time

```
strftime(ts, 256, "%Z", now);
if (strlen(ts) == 0)
    printf("The time zone cannot be determined\n");
else
    printf("The time zone is %s\n", ts);

return 0;
}
```

Results

```
Today's abr.name is Wed
Today's full name is Wednesday
Today's aabr.month name is Apr
Today's full month name is April
Today's date and time is Wednesday April 05 10:50:44 2000
The day of the month is 05
The 24-hour clock hour is 10
The 12-hour clock hour is 10
Today's day number is 096
Today's month number is 04
The minute is 50
The AM/PM is am
The second is 44
The week number of the year, starting on a Sunday is 14
The number of the week is 3
The week number of the year, starting on a Monday is 14
The date is Wednesday April 05 2000
The time is 10:50:44
The last two digits of the year are 00
The year is 2000
The time zone cannot be determined
```

time

Return the current system calendar time.

```
#include <time.h>

time_t time(time_t *timer);
```

timer	time_t *	The address of the time_t variable
-------	----------	------------------------------------

Remarks

The `time()` function returns the computer system's calendar time. If `timer` is not a null pointer, the calendar time is also assigned to the item it points to.

`time()` returns the system current calendar time.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 38.9 Example of `time()` usage.

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

int main(void)
{
    time_t systime;
    systime = time(NULL);

    puts(ctime(&systime));

    return 0;
}
```

Output:

Tue Nov 30 13:06:47 1993

tzset

The function `tzset()` reads the value of the “TZ” environment variable and internalizes it into the time zone functionality of the program.

```
#include <time.h>
void tzset(void);
```

Remarks

The function `tzset()` reads the value of the “TZ” environment variable and internalizes it into the time zone functionality of the program.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“tzname” on page 550](#)

Non Standard <time.h> Functions

Various non standard functions are included in the header `time.h` for legacy source code and compatibility with operating system frameworks and application programming interfaces.

- For the function `strdate` see [“strdate” on page 115](#), for a full description
- [“asctime_r” on page 552](#) is a reentrant version of `asctime`
- [“ctime_r” on page 556](#) is a reentrant version of `ctime`
- [“gmtime_r” on page 559](#) is a reentrant version of `gmtime`
- [“localtime_r” on page 561](#) is a reentrant version of `localtime`

timeb.h

The `timeb.h` header file provides access to the computer system clock, date and time conversion functions, and time formatting functions. Currently this header is implemented for Windows only.

Overview of `timeb.h`

The `timeb.h` facilities include:

- [“`struct timeb`” on page 571](#) lists the elements of the `timeb` structure
- [“`_ftime`” and “`ftime`” on page 572](#) stores the current time in a buffer that the programmer can allocate.

This struct is Windows x86 only at this time. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a compatibility list.

NOTE If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

struct `timeb`

The `timeb` struct is used to store the time of a milliseconds timer, timezone and timezone flag.

Table 39.1 Timeb Structure Members

Field	Description
time_t time	A type used to represent calendar date and time. See “ Type time_t ” on page 548
unsigned short millitm	Current time in milliseconds
short timezone	The difference, in minutes, between local time and Greenwich Mean time
short dstflag	True if daylight savings time is in effect

Remarks

The dstflag flag is true if the daylight savings time is in effect

This structure may not be implemented on all platforms.

ftime

The function ftime and _ftime gets the current time and stores it in a struct that is allocated by the programmer.

```
#include <timeb.h>
void ftime(struct timeb * timebptr);
void _ftime(struct timeb * timebptr);
```

timebptr	struct timeb *	A pointer to a timeb structure that holds the time information
----------	----------------	--

Remarks

The elements of the struct timeb are listed in “[Timeb Structure Members](#)” on page 572

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“ctime”](#) on page 555.

Listing 39.1 Example of ftime usage

```
#include <sys\timeb.h>
#include <stdio.h>

int main()
{
    struct timeb tbuf;
    ftime( &tbuf);

    printf("Time is %s", ctime(&tbuf.time));

    return 0;
}
```

timeb.h

Overview of timeb.h

unistd.h

The header file `unistd.h` contains several functions that are useful for porting a program from UNIX.

Overview of `unistd.h`

The header file `unistd.h` contains several functions that are useful for porting a program from UNIX. These functions are similar to the functions in many UNIX libraries. However, since the UNIX and other operating systems have some fundamental differences, they cannot be identical. The descriptions of the functions tell you what the differences are.

These facilities in `unistd.h` are:

- [“access” on page 576](#) determines the files accessibility.
- [“chdir” on page 577](#) change the directory
- [“close” on page 579](#) close a file opened with open
- [“cuserid” on page 583](#) retrieves the current user’s ID
- [“cwait” on page 584](#) Wait for a process to end.
- [“dup” on page 584](#) duplicates a file handle
- [“dup2” on page 585](#) duplicates to an existing file handle
- [“exec functions” on page 586](#) executes programs from within a program
- [“getcwd” on page 588](#) gets the current working directory
- [“getlogin” on page 589](#) returns a login name
- [“getpid” on page 590](#) returns the process ID
- [“isatty” on page 591](#) determines if a file ID is attached to a terminal
- [“lseek” on page 594](#) seek when opened with open
- [“read” on page 595](#) read when opened with open
- [“rmdir” on page 596](#) removes a directory or folder
- [“sleep” on page 598](#) pauses a program

-
- “[spawn functions](#)” on page 600 spawns a child process
 - “[ttynname](#)” on page 601 determines a terminal id
 - “[unlink](#)” on page 602 deletes a file
 - “[write](#)” on page 604 write to a binary file stream

unistd.h and UNIX compatibility

Generally, you don’t want to use these functions in new programs. Instead, use their counterparts in the native API.

NOTE If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

access

Determines the files accessibility.

```
#include <unistd.h>
int access(const char *fname, int mode);
```

fname	const char *	The file to check
mode	int	The file mode to test for

Remarks

If the access is allowed then zero is returned. A negative number is returned if the access is not allowed.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

The “[Access Test Modes](#)” lists the file modes that may be tested.

Table 40.1 Access Test Modes

F_OK	Test for existence of file
R_OK	Test for read permission
W_OK	Test for write permission
X_OK	Test for execute permission

See Also

[“creat, _wcreate” on page 140](#)

[“open, _wopen” on page 143](#)

[“close” on page 579](#)

chdir

Change the current directory.

```
#include <unistd.h>
int chdir(const char *path);
int _chdir(const char *path);
```

path	char *	The new pathname
------	--------	------------------

Remarks

The function `chdir()` is used to change from one directory to a different directory or folder. Example of usage is given in [“Example of chdir\(\) usage.” on page 578](#)

`chdir()` returns zero, if successful. If unsuccessful `chdir()` returns negative one and sets `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“errno” on page 95](#)

Listing 40.1 Example of chdir() usage.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stat.h>

#define SIZE FILENAME_MAX
#define READ_OR_WRITE 0x0 /* fake a UNIX mode */

int main(void)
{
char folder[SIZE];
char curFolder[SIZE];
char newFolder[SIZE];
int folderExisted = 0;

/* Get the name of the current folder or directory */
getcwd( folder, SIZE );
printf("The current Folder is: %s", folder);

/* create a new sub folder */
/* note mode parameter ignored on Mac */
sprintf(newFolder,"%s%s", folder, "Sub" );
if( mkdir(newFolder, READ_OR_WRITE ) == -1 )
{
    printf("\nFailed to Create folder");
    folderExisted = 1;
}

/* change to new folder */
if( chdir( newFolder) )
{
    puts("\nCannot change to new folder");
    exit(EXIT_FAILURE);
}

/* show the new folder or folder */
getcwd( curFolder, SIZE );
printf("\nThe current folder is: %s", curFolder);

/* go back to previous folder */
if( chdir(folder) )
{
    puts("\nCannot change to old folder");
    exit(EXIT_FAILURE);
}
```

```
}

/* show the new folder or folder */
getcwd( curFolder, SIZE );
printf("\nThe current folder is again: %s", curFolder);

if (!folderExisted)
{
/* remove newly created directory */
if (rmdir(newFolder))
{
    puts("\nCannot remove new folder");
    exit(EXIT_FAILURE);
}
else
    puts("\nNew folder removed");

/* attempt to move to non-existant directory */
if (chdir(newFolder))
    puts("Cannot move to non-existant folder");
else
    puts("\nPre-existing folder not removed");

return 0;
}
```

Output

```
The current Folder is: Macintosh HD:C Reference:  
The current folder is: Macintosh HD:C Reference:Sub:  
The current folder is again: Macintosh HD:C Reference:  
New folder removed  
Cannot move to non-existant folder  
For Windows, refer to "For example of rmdir\(\) usage" on page 597
```

close

Close an open file.

```
#include <unistd.h>

int close(int fildes);
```

fildes	int	The file descriptor
--------	-----	---------------------

Remarks

The `close()` function closes the file specified by the argument. This argument is the value returned by `open()`. Example of usage is given in “[Example of `close\(\)` usage.](#)” on page 581

If successful, `close()` returns zero. If unsuccessful, `close()` returns negative one and sets `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“open, `wopen`” on page 143](#)

[“fclose” on page 346](#)

[“errno” on page 95](#)

Listing 40.2 Example of close() usage.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#define SIZE FILENAME_MAX
#define MAX 1024

char fname[SIZE] = "DonQ.txt";

int main(void)
{
    int fdes;
    char temp[MAX];
    char *Don = "In a certain corner of la Mancha, the name of\n\
which I do not choose to remember,...";
    char *Quixote = "there lived\\none of those country\\
gentlemen, who adorn their\\nhalls with rusty lance\\
and worm-eaten targets.";

    /* NULL terminate temp array for printf */
    memset(temp, '\0', MAX);

    /* open a file */
    if((fdes = open(fname, O_RDWR | O_CREAT ))== -1)
    {
        perror("Error ");
        printf("Can not open %s", fname);
        exit( EXIT_FAILURE );
    }

    /* write to a file */
    if( write(fdes, Don, strlen(Don)) == -1)
    {
        printf("%s Write Error\n", fname);
        exit( EXIT_FAILURE );
    }

    /* move back to over write ... characters */
    if( lseek( fdes, -3L, SEEK_CUR ) == -1L)
    {
```

unistd.h

Overview of unistd.h

```
printf("Seek Error");
exit( EXIT_FAILURE );
}

/* write to a file */
if( write(fdes, Quixote, strlen(Quixote)) == -1)
{
    printf("Write Error");
    exit( EXIT_FAILURE );
}

/* move to beginning of file for read */
if( lseek( fdes, 0L, SEEK_SET ) == -1L)
{
    printf("Seek Error");
    exit( EXIT_FAILURE );
}

/* read the file */
if( read( fdes, temp, MAX ) == 0)
{
    printf("Read Error");
    exit( EXIT_FAILURE );
}

/* close the file */
if(close(fdes))
{
    printf("File Closing Error");
    exit( EXIT_FAILURE );
}

puts(temp);

return 0;
}
```

Result

In a certain corner of la Mancha, the name of which I do not choose to remember, there lived one of those country gentlemen, who adorn their halls with rusty lance and worm-eaten targets.

cuserid

Retrieve the current user's ID.

```
#include <unistd.h>
char *cuserid(char *string);
```

string	char *	The user ID as a string
--------	--------	-------------------------

Remarks

The function `cuserid()` returns the user name associated with the current process. If the string argument is NULL, the file name is stored in an internal buffer. If it is not NULL, it must be at least `FILENAME_MAX` large. Example of usage is given in “[Example of cuserid\(\) usage.” on page 583](#)

For the MacOS, the login name is returned.

`cuserid()` returns a character pointer to the current user's ID.

For the MacOS, the users name is set using the sharing control panel

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 40.3 Example of cuserid() usage.

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

int main(void)
{
    char *c_id = NULL;
    printf("The current user ID is %s\n", cuserid(c_id));

    return 0;
}
```

Result

The current user ID is Metrowerks

cwait

Wait for a process to terminate

```
#include <unistd.h>

int cwait(int *termstat, int pid, int action);
int _cwait(int *termstat, int pid, int action);
```

termstat	int	The termination status
pid	int	Process ID
action	int	The action is ignored

Remarks

The process exit code is returned.

Windows compatible function.

See Also

[“exec functions” on page 586](#)

[“spawn functions” on page 600](#)

dup

Duplicates a file handle.

```
#include <io.h>

int dup(int _a )
int _dup(int _a )
```

_a	int	A file handle to duplicate
----	-----	----------------------------

Remarks

Creates a new file handle for an open file with the same attributes as the original file handle.

A new file handle is returned upon success or a negative one otherwise.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“dup2” on page 585](#)

dup2

Duplicate a file handle unto an existing handle.

```
#include <io.h>
int dup2(int _a, int _b
int _dup2(int _a, int _b
```

_a	int	A file handle to duplicate
_b	int	An existing file handle

Remarks

Associates a file handle for an open file with the same attributes as the original file handle. If the file associated with the second argument is open when _dup2 is called, the old file is closed.

Zero is returned upon success and a negative one upon failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“dup” on page 584](#)

exec functions

Load and execute a child process within the current program memory.

The suffix on the `exec` name determines the method that the child process operates.

- P will search the path variable
- L is used for known argument lists number
- V is for an unknown argument list number
- E allows you to alter an environment for the child process

exec

```
#include <unistd.h>
int exec(const char *path, ...);
int _exec(const char *path, ...);
```

execl

```
int execl(const char *path, ...);
int _execl(const char *path, ...);
```

execle

```
int execle(const char *path, ...);
int _execle(const char *path, ...);
```

execlp

```
int execlp(const char *path, ...);
int _execlp(const char *path, ...);
```

execv

```
int execv(const char *path, ...);
int _execv(const char *path, ...);
```

execve

```
int execve(const char *path, ...);  
int _execve(const char *path, ...);
```

execlp

```
int execlp(const char *path, ...);  
int _execlp(const char *path, ...);
```

NOTE

For the MacOS, all exec-type calls pass through `exec()`, because argument passing (`argc`, `argv`) doesn't exist for MacOS applications

path	const char *	The commandline pathname to execute
...		A variable list of arguments

Remarks

Launches the application named and then quits upon successful launch. Example of usage is given in [“Example of exec\(\) usage.” on page 588.](#)

If successful `exec()` returns zero. If unsuccessful `exec()` returns negative one and sets `errno` according to the error.

For the MacOS using SIOUX, these settings will automatically close the SIOUX program. The `asktosaveonclose` is kept at the default value to demonstrate that the original `printf()` statement is called however the second `printf` statement is not called.

These functions may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Overview of SIOUX” on page 291](#)

[“Overview of errno.h” on page 95](#)

unistd.h

Overview of unistd.h

Listing 40.4 Example of exec() usage.

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

#define SIZE FILENAME_MAX
char appName[SIZE] = "Macintosh HD:SimpleText";

int main(void)
{
    SIOUXSettings.autocloseonquit = 1;
    SIOUXSettings.asktosaveonclose = 1;

    printf("Original Program\n");
    exec(appName);
    printf("program terminated"); /* not displayed */

    return 0;
}

result
Display "Original Program"
after the close of the program
the SimpleText application is launched
```

getcwd

Get the current directory.

```
#include <unistd.h>

char *getcwd(char *buf, int size);
char *_getcwd(char *buf, int size);
```

buf	char	A buffer to hold the pathname of the current working directory
size	int	The size of the buffer

Remarks

The function `getcwd()` takes two arguments. One is a buffer large enough to store the full directory pathname, the other argument is the size of that buffer.

If successful, `getcwd()` returns a pointer to the buffer. If unsuccessful, `getcwd()` returns `NULL` and sets `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Overview of errno.h” on page 95](#)

For example of `getcwd` usage refer to [“Example of chdir\(\) usage.” on page 578](#).

getlogin

Retrieve the username that started the process.

```
#include <unistd.h>
char *getlogin(void);
```

Remarks

The function `getlogin()` retrieves the name of the user who started the program. Example of usage is given in [“Example of getlogin\(\) usage.” on page 590](#)

The Mac doesn't have a login, so this function returns the Owner Name from the File Sharing Setup Control Panel

`getlogin()` returns a character pointer.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

unistd.h

Overview of unistd.h

Listing 40.5 Example of getlogin() usage.

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

int main(void)
{
    printf("The login name is %s\n", getlogin());

    return 0;
}
```

result
The login name is Metrowerks

getpid

Retrieve the process identification number.

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

Listing 40.6 getpid() Macros

Macro	Represents
#define getpid()	Process ID
#define getppid()	Parent process ID
#define getuid()	Real user ID
#define geteuid()	Effective user ID
#define getgid()	Real group ID
#define getegid()	Effective group ID
#define getpgrp()	Process group ID

Remarks

The `getpid()` function returns the unique number (Process ID) for the calling process. Example of usage is given in [“Example of getpid\(\) usage.” on page 591](#)

`getpid()` returns an integer value.

These various related `getpid()` type functions don't really have any meaning on the Mac. The values returned are those that would make sense for a typical user process under UNIX.

`getpid()` always returns a value. There is no error returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 40.7 Example of getpid() usage.

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

int main(void)
{
    printf("The process ID is %d\n", getpid());

    return 0;
}
```

Result
The process ID is 9000

isatty

Determine a specified file_id

```
#include <unistd.h>

int isatty(int fildes);
int _isatty(int fildes);
```

fildes	int	The file descriptor
--------	-----	---------------------

Remarks

The function `isatty()` determines if a specified `file_id` is attached to the console, or if re-direction is in effect. Example of usage is given in [“Example of isatty\(\) ttyname\(\) usage.” on page 593](#)

`isatty()` returns a non-zero value if the file is attached to the console. It returns zero if Input/Output redirection is in effect.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ccommand” on page 56](#)

Listing 40.8 Example of isatty() ttynname() usage.

```
#include <console.h>
#include <stdio.h>
#include <unistd.h>
#include <unix.h>

int main(int argc, char **argv)
{
    int i;
    int file_id;
    argc = ccommand(&argv);

    file_id = isatty(fileno(stdout));
    if(!file_id )
    {
        for (i=0; i < argc; i++)
            printf("command line argument [%d] = %s\n",
                   i, argv[i]);
    }
    else printf("Output to window");

    printf("The associated terminal is %s",
           ttynname(file_id) );

    return 0;
}
```

Result if file redirection is chosen using the command line arguments
Metrowerks CodeWarrior.

Written to file selected:

```
command line argument [0] = CRef
command line argument [1] = Metrowerks
command line argument [2] = CodeWarrior
The associated terminal is SIOUX
```

Iseek

Seek a position on a file stream.

```
#include <unistd.h>
long lseek(int fildes, long offset, int origin);
```

fildes	int	The file descriptor
offset	long	The offset to move in bytes
origin	int	The starting point of the seek

Remarks

The function `lseek()` sets the file position location a specified byte offset from a specified initial location.

The origin of the offset must be one of the positions in [“The lseek offset positions.” on page 594](#).

Listing 40.9 The lseek offset positions.

Macro	Meaning
SEEK_SET	Beginning of file
SEEK_CUR	Current Position
SEEK_END	End of File

If successful, `lseek()` returns the absolute offset as the number of bytes from the beginning of the file after the seek has occurred. If unsuccessful, it returns a value of negative one long integer.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSI/Compatibility> for a Compatibility list.

See Also

[“fseek” on page 387](#)

[“ftell” on page 391](#)

[“open, wopen” on page 143](#)

For example of lseek() usage refer to [“Example of close\(\) usage.” on page 581.](#)

read

Read from a file stream that has been opened in binary mode for unformatted Input/Output.

```
#include <unistd.h>
int read(int fildes, char *buf, int count);
```

fildes	int	The file descriptor
buf	char *	A buffer to store the data read
count	int	The maximum size in bytes to read

Remarks

The function `read()` copies the number of bytes specified by the `count` argument, from the file to the character buffer. File reading begins at the current position. The position moves to the end of the read position when the operation is completed.

NOTE

This function should be used in conjunction with `unistd.h:write()`, and `fcntl.h:open()` only.

`read()` returns the number of bytes actually read from the file. In case of an error a value of negative one is returned and `errno` is set.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fread” on page 375](#)

[“open, wopen” on page 143](#)

For example of `read()` usage refer to [“Example of close\(\) usage.” on page 581.](#)

rmdir

Delete a directory or folder.

```
#include <unistd.h>
int rmdir(const char *path);
```

path	const char *	The pathname of the directory being removed
------	--------------	---

Remarks

The `rmdir()` function removes the directory (folder) specified by the argument.

If successful, `rmdir()` returns zero. If unsuccessful, `rmdir()` returns negative one and sets `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“mkdir” on page 311](#)

[“errno” on page 95](#)

Listing 40.10 For example of rmdir() usage

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stat.h>

#define SIZE FILENAME_MAX
#define READ_OR_WRITE 0x0 /* fake a UNIX mode */

int main(void)
{
char folder[SIZE];
char curFolder[SIZE];
char newFolder[SIZE];
int folderExisted = 0;

/* Get the name of the current folder or directory */
getcwd( folder, SIZE );
printf("The current Folder is: %s", folder);

/* create a new sub folder */
/* note mode parameter ignored on Mac */
sprintf(newFolder,"%s%s", folder, ".\\Sub" );
if( mkdir(newFolder, READ_OR_WRITE ) == -1 )
{
    printf("\nFailed to Create folder");
    folderExisted = 1;
}

/* change to new folder */
if( chdir( newFolder) )
{
    puts("\nCannot change to new folder");
    exit(EXIT_FAILURE);
}

/* show the new folder or folder */
getcwd( curFolder, SIZE );
printf("\nThe current folder is: %s", curFolder);

/* go back to previous folder */
if( chdir(folder) )
{
    puts("\nCannot change to old folder");
    exit(EXIT_FAILURE);
}
```

unistd.h

Overview of unistd.h

```
}

/* show the new folder or folder */
getcwd( curFolder, SIZE );
printf("\nThe current folder is again: %s", curFolder);

if (!folderExisted)
{
/* remove newly created directory */
if (rmdir(newFolder))
{
    puts("\nCannot remove new folder");
    exit(EXIT_FAILURE);
}
else
    puts("\nNew folder removed");

/* attempt to move to non-existant directory */
if (chdir(newFolder))
    puts("Cannot move to non-existant folder");
}
else
    puts("\nPre-existing folder not removed");

return 0;
}

Output
The current Folder is: C:\Programming\CW\Console
The current folder is: C:\Programming\CW\Console\Sub
The current folder is again: C:\Programming\CW\Console
New folder removed
Cannot move to non-existant folder
For Macintosh, refer to "Example of chdir\(\) usage." on page 578.
```

sleep

Delay program execution for a specified number of seconds.

```
#include <unistd.h>
unsigned int sleep(unsigned int sleep);
```

sleep	unsigned int	The length of time in seconds
-------	--------------	-------------------------------

Remarks

The function `sleep()` delays execution of a program for the time indicated by the unsigned integer argument. For the Macintosh system there is no error value returned. Example of usage is given in [“Example of sleep\(\) usage.” on page 599](#)

The function `sleep()` returns zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Listing 40.11 Example of sleep() usage.

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

int main(void)
{
    printf("Output to window\n");
    fflush(stdout); /* needed to force output */

    sleep(3);

    printf("Second output to window");

    return 0;
}
```

Result
Output to window
< there is a delay now >
Second output to window

spawn functions

The `spawn` family of functions create and run other processes (child processes). The suffix on the `spawn` name determines the method that the child process operates.

- P will search the path variable
- L is used for known argument lists number
- V is for an unknown argument list number
- E allows you to alter an environment for the child process

spawnl

```
int spawnl(int,const char *, ...);  
int _spawnl(int,const char *, ...);
```

spawnv

```
int spawnv(int,const char *,const char *const*);  
int _spawnv(int,const char *,const char *const*);
```

spawnle

```
int spawnle(int,const char *,...);  
int _spawnle(int,const char *,...);
```

spawnve

```
int spawnve(int,const char *,const char *const*, const char  
*const*);  
int _spawnve(int,const char *,const char *const*, const char  
*const*);
```

spawnlp

```
int spawnlp(int,const char *,...);  
int _spawnlp(int,const char *,...);
```

spawnvp

```
int spawnvp(int,const char *,const char *const *);  
int _spawnvp(int,const char *,const char *const *);
```

spawnlpe

```
int spawnlpe(int,const char *,...);  
int _spawnlpe(int,const char *,...);
```

spawnvpe

```
int spawnvpe(int,const char *,const char *const *, const char  
*const*);  
int _spawnvpe(int,const char *,const char *const *, const char  
*const*);
```

Remarks

All `spawn` functions take a variable argument list as a parameter.

The child process's exit status is returned.

These functions may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“exec functions” on page 586](#)

ttynname

Retrieve the name of the terminal associated with a file ID.

```
#include <unistd.h>  
char *ttynname(int fildes);
```

fildes	int	The file descriptor
--------	-----	---------------------

Remarks

The function `ttyname()` retrieves the name of the terminal associated with the file ID.

`ttyname()` returns a character pointer to the name of the terminal associated with the file ID, or `NULL` if the file ID doesn't specify a terminal.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

For example of `ttyname()` usage refer to [“Example of `isatty\(\)` `ttyname\(\)` usage.” on page 593](#).

unlink

Delete (unlink) a file.

```
#include <unistd.h>
int unlink(const char *path);
```

path	const char *	A pathname of the file to remove
------	--------------	----------------------------------

Remarks

The function `unlink()` removes the specified file from the directory. Example of usage is given in [“Example of `unlink\(\)` usage.” on page 603](#)

If successful, `unlink()` returns zero. If unsuccessful, it returns a negative one.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“rmdir” on page 596](#)

[“mkdir” on page 311](#)

Listing 40.12 Example of unlink() usage.

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

#define SIZE FILENAME_MAX

int main(void)
{
    FILE *fp;
    char fname[SIZE] = "Test.txt";

    /* create a file */
    if( (fp = fopen( fname, "w" ) ) == NULL )
    {
        printf("Can not open %s for writing", fname);
        exit( EXIT_FAILURE );
    }
    else printf("%s was opened for writing\n", fname);

    /* display it is available */
    if( !fclose(fp) ) printf("%s was closed\n", fname);

    /* delete file */
    if( unlink(fname) )
    {
        printf("%s was not deleted", fname);
        exit( EXIT_FAILURE );
    }

    /* show it can't be re-opened */
    if( (fp = fopen( fname, "r" ) ) == NULL )
    {
        printf("Can not open %s for reading it was deleted",
               fname);
        exit( EXIT_FAILURE );
    }
    else printf("%s was opened for reading\n", fname);

    return 0;
}
```

Result
Test.txt was opened for writing
Test.txt was closed

```
Can not open Test.txt for reading it was deleted
```

write

Write to a file stream that has been opened in binary mode for unformatted Input/Output.

```
#include <unistd.h>
int write(int fildes, const char* buf, size_t count)
```

fildes	int	The file descriptor
buf	const char *	The address of the buffer being written
count	size_t	The size of the buffer being written

Remarks

The function `write()` copies the number of bytes in the `count` argument from the character array buffer to the file `fildes`. The file position is then incremented by the number of bytes written.

This function should be used in conjunction with “[read” on page 595](#), and “[open, wopen” on page 143](#) only.

`write()` returns the number of bytes actually written.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fwrite” on page 394](#)

[“read” on page 595](#)

[“open, wopen” on page 143](#)

For example of `write()` usage refer to [“Example of close\(\) usage.” on page 581](#).

unistd.h

Overview of unistd.h

unix.h

The header file `unix.h` contains two global variables.

Overview of unix.h

The header file `unix.h` contains two global variables that are useful for console interface programs for setting a Mac OS file creator and file type.

The globals variables in `unix.h` are:

- “[_fcreator](#)” on page 607 sets a Macintosh file creator
- “[_ftype](#)” on page 608 sets a Macintosh file type

UNIX Compatibility

Generally, you don’t want to use these functions in new programs. Instead, use their counterparts in the native API.

NOTE

If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

_fcreator

To specify a Macintosh file creator.

```
#include <unix.h>
extern long _fcreator
```

Remarks

Use the global _fcreator to set the creator type for files created using the Standard C Libraries. [“Using global variables to set file creator and type.” on page 609](#) is an example of the use of the global variable _fcreator.

This global identifier is Macintosh Only

_ftype

To specify a Macintosh file type.

```
#include <unix.h>
extern long _ftype;
```

Remarks

Use the global _ftype to set the creator type for files created using the Standard C Libraries. [“Using global variables to set file creator and type.” on page 609](#) is an example of the use of the global variable _ftype.

The value assigned to _fcreate and _ftype is a ResType (i.e. four character constant).

This global identifier is Macintosh Only

Listing 41.1 Using global variables to set file creator and type.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unix.h>

#define oFile "test file"
const char *str = "Metrowerks Software at Work";


int main(void)
{
    FILE *fp;
    _fcreator = 'tttxt';
    _ftype = 'TEXT';

    // create a new file for output and input
    if (( fp = fopen(oFile, "w+")) == NULL)
    {
        printf("Can't create file.\n");
        exit(1);
    }

    fwrite(str, sizeof(char), strlen(str), fp);
    fclose(fp);

    return 0;
}

// output to the file using fwrite()
Metrowerks Software at Work
```

unix.h

Overview of unix.h

utime.h

The header file `utime.h` contains several functions that are useful for porting a program from UNIX.

Overview of utime.h

The header file `utime.h` contains several functions that are useful for porting a program from UNIX. These functions are similar to the functions in many UNIX libraries. However, since the UNIX and Macintosh operating systems have some fundamental differences, they cannot be identical. The descriptions of the functions tell you what the differences are.

The facilities in `utime.h` are:

- [“utime” on page 612](#) to set a file modification time
- [“utimes” on page 615](#) to set a series of file modification times

utime.h and UNIX Compatibility

Generally, you don’t want to use these functions in new programs. Instead, use their counterparts in the native API.

NOTE

If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

utime

Sets a file's modification time.

```
#include <utime.h>

int utime(const char *path, /* Mac */ const struct utimbuf *buf);
int utime(const char *path, /* Windows */ struct utimbuf *buf);
```

path	const char *	The pathname as a string
buf	const struct utimbuf * struct utimbuf	The address of a struct that will hold a file's time information

Remarks

This function sets the modification time for the file specified in *path*. Since the Macintosh does not have anything that corresponds to a file's access time, it ignores the *actime* field in the *utimbuf* structure.

If *buf* is NULL, *utime()* sets the modification time to the current time. If *buf* points to a *utimbuf* structure, *utime()* sets the modification time to the time specified in the *modtime* field of the structure.

The *utimbuf* structures contains the fields in [Listing 42.3](#).

Listing 42.1 The *utimbuf* structure

This field...	is the...
time_t actime	Access time for the file. Since the Macintosh has nothing that corresponds to this, <i>utime()</i> ignores this field.
time_t modtime	The last time this file was modified.

If it is successful, *utime()* returns zero. If it encounters an error, *utime()* returns -1 and sets *errno*.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ctime” on page 555](#)

[“mktime” on page 561](#)

[“fstat” on page 309](#)

[“stat” on page 312](#)

[“utimes” on page 615](#)

Listing 42.2 Example for utime()

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

int main(void)
{
    struct utimbuf timebuf;
    struct tm date;
    struct stat info;
    FILE * fp;

    fp = fopen("mytest", "w");
    if(!fp)
    {
        fprintf(stderr, "Could not open file");
        exit(EXIT_FAILURE);
    }
    fprintf(fp, "test");
    fclose(fp);
    /* Create a calendar time for
    Midnight, Apr. 4, 1994. */
    date.tm_sec=0;      /* Zero seconds      */
    date.tm_min=0;      /* Zero minutes     */
    date.tm_hour=0;     /* Zero hours       */
    date.tm_mday=4;     /* 4th day          */
    date.tm_mon=3;      /* .. of April      */
    date.tm_year=94;    /* ...in 1994       */
    date.tm_isdst=-1;   /* Not daylight savings */
    timebuf.modtime=mktime(&date);
    /* Convert to calendar time.      */

    /* Change modification date to  *
     * midnight, Apr. 4, 1994.      */
    utime("mytest", &timebuf);
    stat("mytest", &info);
    printf("Mod date is %s", ctime(&info.st_mtime));

    /* Change modification date *
     * to current time          */
    utime("mytest", NULL);
    stat("mytest", &info);
    printf("Mod date is %s", ctime(&info.st_mtime));

    return 0;
}
```

}

This program might display the following to standard output:

Mod date is Mon Apr 4 00:00:00 1994
Mod date is Mon Jul 10 17:45:17 2000

utimes

Sets a file's modification time

```
#include <utime.h>
int utimes(const char *path, struct timeval buf[2]);
```

path	const char *	The pathname as a string
buf	timeval struct array	An array of time values used to set the modification dates

Remarks

This function sets the modification time for the file specified in path to the second element of the array buf. Each element of the array buf is a timeval structure, which has the fields in [Listing 42.3](#).

Listing 42.3 The timeval structure

This field	is the
int t	tv_sec
int	tv_usec

The first element of buf is the access time.

Since the Macintosh does not have anything that corresponds to a file's access time, it ignores that element of the array.

If it is successful, utimes () returns 0. If it encounters an error, utimes () returns -1 and sets errno.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“utime” on page 612](#)

[“ctime” on page 555](#)

[“mktime” on page 561](#)

[“fstat” on page 309](#)

[“stat” on page 312](#)

Listing 42.4 Example for utimes()

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

int main(void)
{
    struct tm date;
    struct timeval buf[2];
    struct stat info;

    /* Create a calendar time for
    Midnight, Sept. 9, 1965.*/
    date.tm_sec=0;          /* Zero seconds */
    date.tm_min=0;          /* Zero minutes */
    date.tm_hour=0;         /* Zero hours */
    date.tm_mday=9;         /* 9th day */
    date.tm_mon=8;          /* .. of September */
    date.tm_year=65;         /* ...in 1965 */
    date.tm_isdst=-1;        /* Not daylight savings */
    buf[1].tv_sec=mktime(&date);
    /* Convert to calendar time. */

    /* Change modification date to      *
     * midnight, Sept. 9, 1965.          */
    utimes("mytest", buf);
    stat("mytest", &info);
    printf("Mod date is %s", ctime(&info.st_mtime));

    return 0;
}
```

This program prints the following to standard output:
Mod date is Thu Sep 9 00:00:00 1965

utime.h

Overview of utime.h

utsname.h

The header file `utsname.h` contains several functions that are useful for porting a program from UNIX.

Overview of `utsname.h`

The header file `utsname.h` contains several functions that are useful for porting a program from UNIX. These functions are similar to the functions in many UNIX libraries. However, since the UNIX and Macintosh operating systems have some fundamental differences, they cannot be identical. The descriptions of the functions tell you what the differences are.

The header `utsname.h` has one function [“`uname`” on page 620](#) that retrieves information on the system you are using.

`utsname.h` and UNIX Compatibility

Generally, you don’t want to use these functions in new programs. Instead, use their counterparts in the Macintosh Toolbox.

NOTE	If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.
-------------	---

See Also

[“MSL Extras Library Headers” on page 30](#) for information on POSIX naming conventions.

utsname.h

Overview of utsname.h

uname

Gets information about the system you are using.

```
#include <utsname.h>
int uname(struct utsname *name);
```

name	struct utsname *	A struct to store system information
------	------------------	--------------------------------------

Remarks

This function gets information on the Macintosh you’re using and puts the information in the structure that `name` points to. The structure contains the fields listed in [Listing 43.1](#). All the fields are null-terminated strings.

Listing 43.1 The utsname structure

This field...	is...
sysnam	The operating system
nodename	The sharing node name.
release	The release number of system software.
version	The minor release numbers of the system software version.
machine	The type of the machine that you are using.

If it is successful, `uname()` returns zero. If it encounters an error, `uname()` returns `-1` and sets `errno`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“fstat” on page 309](#)

[“stat” on page 312](#)

Listing 43.2 Example of uname() usage.

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

int main(void)
{
    struct utsname name;

    uname(&name);

    printf("Operating System: %s\n", name.sysname);
    printf("Node Name:          %s\n", name.nodename);
    printf("Release:            %s\n", name.release);
    printf("Version:            %s\n", name.version);
    printf("Machine:            %s\n", name.machine);

    return 0;
}
```

This application could print the following:

Operating System: MacOS
Node Name: Ron's G4
Release: 9
Version: 3
Machine: Power Macintosh

This machine is a Power Macintosh running Version 9 of the MacOS. The Macintosh Name field of the File Sharing control panel contains "Ron's G4"

utsname.h

Overview of utsname.h

wchar.h

The header file wchar.h contains defines and functions to manipulate wide character sets.

Overview of wchar.h

The header `wchar.h` has many diverse functions and definitions.

Input and output facilities

- [“fgetwc” on page 628](#) behaves like `fgetc` for wide character arrays
- [“fgetws” on page 629](#) behaves like `fgets` for wide character arrays
- [“fputwc” on page 630](#) behaves like `fputc` for wide character arrays
- [“fputws” on page 630](#) behaves like `fputs` for wide character arrays
- [“fwprintf” on page 631](#) behaves like `fprintf` for wide character arrays
- [“fwscanf” on page 632](#) behaves like `fscanf` for wide character arrays
- [“getwc” on page 633](#) behaves like `getc` for wide character arrays
- [“getwchar” on page 634](#) behaves like `getchar` for wide character arrays
- [“putwc” on page 638](#) behaves like `putc` for wide character arrays
- [“putwchar” on page 639](#) behaves like `putchar` for wide character arrays
- [“swprintf” on page 640](#) behaves like `sprintf` for wide character arrays
- [“swscanf” on page 641](#) behaves like `sscanf` for wide character arrays
- [“wprintf” on page 676](#) behaves like `printf` for wide character arrays
- [“wscanf” on page 679](#) behaves like `scanf` for wide character arrays
- [“vfwprintf” on page 644](#), behaves like `vprintf` for wide character arrays
- [“vfscanf” on page 641](#) behaves like `vfscanf` for wide character arrays
- [“vswscanf” on page 642](#) behaves like `vsscanf` for wide character arrays
- [“vwprintf” on page 647](#) behaves like `vprintf` for wide character arrays

- “[vswprintf](#)” on page 646 behaves like `fgetwc` `vsprintf` for wide character arrays
- “[vwscanf](#)” on page 643 behaves like `vscanf` for wide character arrays

Time facilities

- “[wcsftime](#)” on page 653 behaves like `csftime` for wide character arrays
- “[wctime](#)” on page 671 behaves like `ctime` for wide character arrays

String facilities

- “[watof](#)” on page 648 behaves like `atof` for wide characters array
- “[wcscat](#)” on page 649 behaves like `strcat` for wide character arrays
- “[wcschr](#)” on page 650 behaves like `strchr` for wide character arrays
- “[wcscmp](#)” on page 650 behaves like `strcmp` for wide character arrays
- “[wcscspn](#)” on page 652 behaves like `strspn` for wide character arrays
- “[wcscoll](#)” on page 651 behaves like `strcoll` for wide character arrays
- “[wcscopy](#)” on page 652 behaves like `strcpy` for wide character arrays
- “[wcslen](#)” on page 654 behaves like `strlen` for wide character arrays
- “[wcsncat](#)” on page 654 behaves like `strncat` for wide character arrays
- “[wcsncmp](#)” on page 655 behaves like `strncmp` for wide character arrays
- “[wcsncpy](#)” on page 656 behaves like `strncpy` for wide character arrays
- “[wcspbrk](#)” on page 657 behaves like `strbrk` for wide character arrays
- “[wcsrchr](#)” on page 657 behaves like `strrchr` for wide character arrays
- “[wcsspn](#)” on page 659 behaves like `strspn` for wide character arrays
- “[wcsstr](#)” on page 660 behaves like `strstr` for wide character arrays
- “[wcstod](#)” on page 660 behaves like `strtod` for wide character arrays
- “[wcstof](#)” on page 661 behaves like `strtod` for wide character arrays
- “[wcstok](#)” on page 663 behaves like `strtok` for wide character arrays
- “[wcstol](#)” on page 664 behaves like `strtol` for wide character arrays
- “[wcstold](#)” on page 665 behaves like `strtold` for wide character arrays
- “[wcstoll](#)” on page 666 behaves like `strtoll` for wide character arrays
- “[wcstoul](#)” on page 667 behaves like `strtoul` for wide character arrays
- “[wcstoull](#)” on page 669 behaves like `strtoull` for wide character arrays

- [“wcsxfrm” on page 670](#) behaves like `strxfrm` for wide character arrays
- [“wmemchr” on page 672](#) behaves like `memchr` for wide character arrays
- [“wmemcpy” on page 674](#) behaves like `memcpy` for wide character arrays
- [“wmemcmp” on page 673](#) behaves like `memcmp` for wide character arrays
- [“wmemmove” on page 674](#) behaves like `memmove` for wide character arrays
- [“wmemset” on page 675](#) behaves like `memset` for wide character arrays

Multibyte character functions

- [“mbrlen” on page 634](#) behaves like `strlen` for multibyte character arrays
- [“mbrtowc” on page 635](#) converts multibyte characters to wide character arrays
- [“mbsinit” on page 637](#) determines the multibyte conversion status
- [“mbsrtowcs” on page 637](#) converts multibyte strings to wide character strings
- [“wcrtomb” on page 648](#) converts wide characters to multibyte character arrays.
- [“wcsrtombs” on page 658](#) converts wide character strings to multibyte strings.

Conversion functions

- [“btowc” on page 628](#) converts byte characters to wide character arrays
- [“wctob” on page 671](#) converts wide characters to byte character arrays

Wide Character and Byte Character Stream Orientation

There are two types of stream orientation for input and output, a wide character (`wchar_t`) oriented and a byte (`char`) oriented. A stream is un-oriented after that stream is associated with a file, until a byte or wide character input/output operation occurs.

Once any input/output operation is performed on that stream, that stream becomes oriented by that operation to be either byte oriented or wide character oriented and remains that way until the file has been closed and reopened.

Listing 44.1 The Byte Oriented Functions in Stdio.h are:

fgetc	fgets	fprintf	fputc	fputs
fread	fscanf	fwrite	getc	getchar
gets	printf	putc	putchar	puts
scanf	ungetc	vfprintf	vfscanf	vprintf

Listing 44.2 The wide character Oriented Functions are:

fgetwc	fgetws	fwprintf	fputwc	fputws	fwscanf
getwc	getwchar	putwc	putwchar	swprintf	swscanf
towctrans	vfwscanf	vswscanf	vwscanf	vfwprintf	vswprintf
vwprintf	wasctime	watof	wcscat	wcschr	wcscmp
wcscoll	wcscspn	wcscopy	wcslen	wcsncat	wcsncmp
wcsncpy	wcspbrk	wcsspn	wcsrchr	wcsstr	wcstod
wcstok	wcsftime	wcsxfrm	wctime	wctrans	wmemchr
wmemcmp	wmemcpy	wmemmove	wmemset	wprintf	wscanf

After a stream orientation is established, any call to an input/output function of the other orientation is not applied. For example, a byte-oriented input/output function does not have an effect on a wide-oriented stream.

Unicode

Unicode, also known as UCS-2 (Universal Character Set containing 2 bytes) is a fixed-width encoding scheme that uses 16 bits per character. Characters are represented and manipulated in MSL as wide characters of type `wchar_t` and can be manipulated with the wide character functions defined in the C Standard.

Multibyte characters

A Unicode character may be encoded as a sequence of one or more 8-bit characters, this is a multibyte character. There are two types of multibyte encoding, modal and non-modal. With modal encoding, a conversion state is associated with a multibyte string; this state is akin to the shift state of a keyboard. With non-modal encoding, no such state is involved and the first character of a multibyte sequence contains

information about the number of characters in the sequence. The actual encoding scheme is defined in the LC_CTYPE component of the current locale.

In MSL, two encoding schemes are available, a direct encoding where only a single byte is used and the non-modal UTF-8 (UCS Transformation Format -8) encoding scheme is used where each Unicode character is represented by one to three 8-bit characters. For Unicode characters in the range 0x0000 to 0x007F the encoding is direct and only a single byte is used.

Stream Orientation and Standard Input/Output

The three predefined associated streams, stdin, stdout, and stderr are un-oriented at program startup. If any of the standard input/output streams are closed it is not possible to reopen and reconnect that stream to the console. However, it is possible to reopen and connect the stream to a named file.

The C and C++ input/output facilities share the same stdin, stdout and stderr streams.

Definitions

The header wchar.h includes specific definitions for use with wide character sets.

Listing 44.3 Wide Character Definitions

Defines	Definitions
mbstate_t	A value that can hold the conversion state for mode-dependent multibyte encoding
WCHAR_MIN	Minimum value of a wide char
WCHAR_MAX	Maximum value of a wide char
WEOF	A value that differs from any member of the wide character set and is used to denote end of file
win_t	An int type that can hold any wide character representation and WEOF

btowc

The function `btowc()` converts a single byte character to a wide character.

```
#include <wchar.h>
wint_t btowc(int c);
```

int	c *	The character to be converted
-----	-----	-------------------------------

Returns

The function `btowc()` returns the wide character representation of the argument or WEOF is returned if `c` has the value EOF or if the current locale specifies that UTF-8 encoding is to be used and unsigned unsigned char `c` does not constitute a valid single-byte UTF-8 encoded character.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wctob” on page 671](#)

fgetwc

Gets a wide character from a file stream.

```
#include <wchar.h>
wchar_t fgetwc(FILE * file);
```

file	FILE *	The input stream to retrieve from
------	--------	-----------------------------------

Remarks

Performs the same task as `fgetc` for wide character

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Returns the wide character or WEOF for an error

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 625](#)
- [“fgetc” on page 356](#)

fgetws

The function `fgetws()` reads a wide character string from a file stream.

```
#include <wchar.h>
wchar_t *fgetws(wchar_t * s, int n, FILE * file);
```

s	wchar_t *	A wide char string to receive input
n	int	Maximum number of wide characters to be read
file	FILE *	A pointer to the input file stream

Remarks

Behaves like `fgets()` for wide characters.

On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Returns a pointer to `s` if successful or NULL for a failure.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 625](#)
- [“fgets” on page 360](#)

fputwc

Inserts a single wide character into a file stream.

```
#include <wchar.h>
wchar_t fputwc(wchar_t c, FILE * file);
```

c	wchar_t	The wide character to insert
file	FILE *	A pointer to the output file stream

Remarks

Performs the same task as `fputc()` for a wide character type.

On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Returns the wide character written if it is successful, and returns `WEOF` if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)

[“fputc” on page 371](#)

fputws

Inserts a wide character array into a file stream

```
#include <wchar.h>
int fputws(const wchar_t * s, FILE * file);
```

s	wchar_t *	The string to insert
file	FILE *	A pointer to the output file stream

Remarks

Performs the same task as `fputs` for a wide character type.

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the `fflush()` function or one of the file positioning operations (`fseek()`, `fsetpos()`, or `rewind()`).

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Returns a zero if successful, and returns a nonzero value when it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 625](#)
- [“fputs” on page 373](#)

fwprintf

Formatted file insertion

```
#include <wchar.h>
int    fwprintf(FILE * file, const wchar_t * format, ...);
```

file	FILE *	A pointer to the output file stream
format	wchar_t *	The format string
....		Variable arguments

Remarks

Performs the same task as `fprintf()` for a wide character type.

The `fwprintf()` function writes formatted text to a wide character stream and advances the file position indicator. Its operation is the same as `wprintf()` with the addition of the `stream` argument.

Refer to the [“wprintf” on page 676](#) function for details of the format string.

On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Returns the number of arguments written or a negative number if an error occurs

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)

[“wprintf” on page 676](#)

fwscanf

Reads formatted text from a stream.

```
#include <wchar.h>
int fwscanf(FILE * file, const wchar_t * format, ...);
```

file	FILE *	The input file stream
format	wchar_t *	The format string
....		Variable arguments

Remarks

Performs the same task as `fscanf` function for the wide character type.

The `fwscanf()` function reads programmer-defined, formatted text from a wide character stream. The function operates identically to the `wscanf()` function with the addition of the `stream` argument indicating the stream to read from.

Refer to the [“wscanf” on page 679](#) function for details of the format string.

NOTE On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Returns the number of items read, which can be fewer than provided for in the event of an early matching failure. If the end of file is reached before any conversions are made, `EOF` is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 625](#)
- [“wscanf” on page 679](#)

getwc

Reads the next wide character from a wide character stream.

```
#include <wchar.h>
wchar_t getwc(FILE * file);
```

file	FILE *	The file stream
------	--------	-----------------

Remarks

Performs the same task as `getc` for a wide character type.

Returns the next wide character from the stream or returns `WEOF` if the end-of-file has been reached or a read error has occurred.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 625](#)
- [“getc” on page 395](#)

getwchar

Returns a wide character type from the standard input.

```
#include <wchar.h>
wchar_t getwchar(void);
```

Has no parameters.

Remarks

Performs the same task as `getchar` for a wide character type.

Returns the value of the next wide character from `stdin` as an `int` if it is successful. `getwchar()` returns `WEOF` if it reaches an end-of-file or an error occurs.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)

[“getwchar” on page 634](#)

mbrlen

Computes the length of a multibyte character encoded as specified in the `LC_CTYPE` component of the current locale. This function is essentially the same as `mblen()` except that it has an additional parameter of type `mbstate_t*`, which is ignored if the encoding scheme is non-modal.

```
#include <stdlib.h>
int mbrlen(const char *s, size_t n, mbstate_t * ps);
```

s	const char **	The multibyte array to measure
---	---------------	--------------------------------

n	size_t	The Maximum size
ps	mbstate_t **	The current state of translation between multibyte and wide character, ignored if the encoding scheme is non-modal.

Remarks

The `mbrlen()` function returns the length of the multibyte character pointed to by `s`. It examines a maximum of `n` characters.

The Metrowerks C implementation supports the "C" locale with UTF-8 encoding only and returns the value of `mbrtowc(NULL, s, n, pc)`.

`mbrlen()` returns the value of `mbrtowc(NULL, s, n, pc)`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wcslen” on page 654](#)

[“strlen” on page 528](#)

mbrtowc

Translate a multibyte character to a `wchar_t` type according to the encoding specified in the `LC_CTYPE` component of the current locale. This function is essentially the same as `mbtowcs()` except that it has an additional parameter of type `mbstate_t*`, which is ignored if the encoding scheme is non-modal.

```
#include <stdlib.h>
int mbrtowc(wchar_t *pwc,
            const char *s, size_t n, mbstate_t * ps);
```

pwc	wchar_t *	The wide character destination
s	const char *	The string to convert

n	size_t	The maximum wide characters to convert
ps	mbstate_t *	The current state of translation between multibyte and wide character, ignored if the encoding scheme is non-modal.

Remarks

If `s` is a null pointer, this call is equivalent to `mbrtowcs(NULL, "", 1, ps);`

If `s` is not a null pointer, the `mbrtowc()` function examines at most `n` bytes starting with the byte pointed to by `s` to determine how many bytes are needed to complete a valid encoding of a Unicode character. If this is less than or equal to `n` and `pwc` is not a null pointer, it converts the multibyte character, pointed to by `s`, to a character of type `wchar_t`, pointed to by `pwc` using the encoding scheme specified in the `LC_CTYPE` component of the current locale.

`mbrtowc()` returns the first of the following values that applies:

Zero, if `s` points to a null character, which is the value stored.

Greater than zero, if `s` points to a complete and valid multibyte character of `n` or fewer bytes, the corresponding Unicode wide character is stored (if `pwc` is not `NULL`) and the value returned is the number of bytes in the complete multibyte character.

(`size_t`) (-2) if the next `n` bytes pointed to by `s` constitute an incomplete but potentially valid multibyte character. No value is stored.

(`size_t`) (-1) if the next `n` or fewer bytes pointed to by `s` do not constitute a complete and valid multibyte character. The value of `EILSEQ` is stored in `errno` but no wide character value is stored.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“mbsrtowcs” on page 637](#)

[“wcrtomb” on page 648](#)

[“wcsrtombs” on page 658](#)

mbsinit

Determines if the multi-byte state is the initial conversion state or not.

```
#include <wchar.h>
int mbsinit(const mbstate_t *ps);
```

ps	const mbstate_t *	A pointer to a mbstate_t object
----	-------------------	---------------------------------

Remarks

If the status of the object pointed to is a null pointer or is in the initial conversion state a true values is retruned otherwise zero is returned.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“mbsrtowcs” on page 637](#)
- [“wcrtomb” on page 648](#)
- [“wcsrtombs” on page 658](#)

mbsrtowcs

Convert a multibyte character array to a wchar_t array. This function is essentially the same as mbstowcs() except that it has an additional parameter of type mbstate_t*, which is ignored if the encoding scheme is non-modal.

```
#include <stdlib.h>
size_t mbsrtowcs(wchar_t *pwcs,
const char **s, size_t n, mbstate_t * ps);
```

pwcs	wchar_t *	The wide character destination
s	const char **	Indirect pointer to the string to convert

n	size_t	The maximum wide characters to convert
ps	mbstate_t *	The current state of translation between multibyte and wide character, ignored if the encoding scheme is non-modal.

Remarks

The Metrowerks implementation of `mbsrtowcs()` converts a sequence of multibyte characters encoded according to the scheme specified in the `LC_CTYPE` component of the current locale from the wide character array indirectly pointed to by `s` and, if `pwcs` is not a null pointer, stores not more than `n` of the corresponding Unicode characters into the wide character array pointed to by `pwcs`. The function terminates prematurely if a null character is reached, in which case the null wide character is stored, or if an invalid multibyte encoding is detected. If conversion stops because a terminating null character is reached, a null pointer is assigned to the object pointed to by `s`, otherwise a pointer to the address just beyond the last multibyte character converted, if any.

If an invalidly encoded wide character is encountered, `mbsrtowcs()` stores the value of `EILSEQ` in `errno` and returns the value `(size_t)(-1)`. Otherwise `mbsrtowcs()` returns the number of multibyte characters successfully converted, not including any terminating null wide character.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wcsrtombs” on page 658](#)

[“mbtowc” on page 635](#)

putwc

Write a wide character type to a stream.

```
#include <wchar.h>
wchar_t putwc(wchar_t c, FILE * file);
```

c	wchar_t	The wide character to output
file	FILE	The output stream

Remarks

Performs the same task as `putc` for a wide character type.

Returns the wide character written when successful and returns `WEOF` when it fails

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)
[“putc” on page 410](#)

putwchar

Writes a wide character to standard output.

```
#include <wchar.h>
wchar_t putwchar(wchar_t c);
```

c	wchar_t	The wide character to write.
---	---------	------------------------------

Remarks

Performs the same task as `putchar` for a wide character type.

Returns `c` if it is successful and returns `WEOF` if it fails

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)
[“putchar” on page 412](#)

swprintf

Formats text in a wide character type string.

```
#include <wchar.h>
int swprintf(wchar_t * s, size_t N, const wchar_t * format, ...);
```

s	wchar_t*	The string buffer to hold the formatted text
n	size_t	Number of characters allowed to be written
format	wchar_t*	The format string
....		Variable arguments

Remarks

Performs the same task as `sprintf` for a wide character type with an additional parameter for the maximum number of wide characters to be written. No more than `n` wide characters will be written including the terminating `NULL` wide character, which is always added unless the `n` is zero.

Refer to the [“wprintf” on page 676](#) function for details of the format string.

Returns the number of characters assigned to `s`, not including the null character, or a This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

negative number if `n` or more characters were requested to be written.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)

[“fwprintf” on page 631](#)

[“sprintf” on page 431](#)

swscanf

Reads a formatted wide character string.

```
#include <wchar.h>
int swscanf(const wchar_t * s, const wchar_t * format, ...);
```

s	wchar_t*	The string being read
format	wchar_t*	The format string
....		Variable arguments

Remarks

Performs the same task as `sscanf` for a wide character type.

Returns the number of items successfully read and converted, which can be fewer than provided for in the event of an early matching failure. If the end of the input string is reached before any conversions are made, EOF is returned.

Refer to the ["wscanf" on page 679](#) function for details of the format string.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

["Wide Character and Byte Character Stream Orientation" on page 625](#)
["scanf" on page 420](#)

vfwscanf

Read formatted text from a wide character stream.

```
#include <wchar.h>
int vfwscanf(FILE * file,
              const wchar_t * format_str, va_list arg);
```

file	FILE *	The stream being read
format_str	const wchar_t*	The format string
....		Variable arguments

Remarks

Performs the same task as `fscanf` for a wide character type.

Refer to the ["wscanf" on page 679](#) function for details of the format string.

NOTE	On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.
-------------	---

`vfwscanf()` returns the number of items assigned, which can be fewer than provided for in the case of an early matching failure. If an input failure occurs before any conversion, `vfwscanf()` returns EOF.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

["Wide Character and Byte Character Stream Orientation" on page 625](#)

["fscanf" on page 379](#)

vswscanf

Reads formatted text from a wide character string.

```
#include <wchar.h>
int __vswscanf(const wchar_t * s,
                const wchar_t * format, va_list arg);
```

s	wchar_t*	The string being read
---	----------	-----------------------

format	wchar_t*	The format string
.arg	va_list	A variable argument list

Remarks

The `vswscanf()` function works identically to the `swscanf()` function. Instead of the variable list of arguments that can be passed to `swscanf()`, `vswscanf()` accepts its arguments in the array `arg` of type `va_list`, which must have been initialized by the `va_start()` macro (and possibly subsequent `va_arg` calls) from the `stdarg.h` header file. The `vswscanf()` function does not invoke the `va_end` macro.

Refer to the [“wscanf” on page 679](#) function for details of the format string.

NOTE

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

`vswscanf()` returns the number of items assigned, which can be fewer than provided for in the case of an early matching failure. If an input failure occurs before any conversion, `vswscanf()` returns `EOF`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)
[“sscanf” on page 432](#)

vwscanf

Reads formatted text from wide character oriented `stdin`.

```
#include <wchar.h>
int vwscanf(const wchar_t * format, va_list arg);
```

s	wchar_t*	The string being read
format	wchar_t*	The format string
....		Variable arguments

Remarks

The `vwscanf()` function works identically to the `wscanf()` function. Instead of the variable list of arguments that can be passed to `wscanf()`, `vwscanf()` accepts its arguments in the array `arg` of type `va_list`, which must have been initialized by the `va_start()` macro (and possibly subsequent `va_arg` calls) from the `stdarg.h` header file. The `vwscanf()` function does not invoke the `va_end` macro.

Refer to the ["wscanf" on page 679](#) function for details of the format string.

The `vwscanf()` function returns the number of items assigned, which can be fewer than provided for in the case of an early matching failure. If an input failure occurs before any conversion, `vwscanf()` returns `EOF`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

["Wide Character and Byte Character Stream Orientation" on page 625](#)

["vfwscanf" on page 641](#)

["scanf" on page 420](#)

vfwprintf

Write a formatted text to a file stream.

```
#include <wchar.h>
int vfwprintf(FILE * file,
const wchar_t * format_str, va_list arg);
```

file	FILE *	The stream being written
format_str	wchar_t*	The format string
arg	va_list	A variable argument list

Remarks

The `vfwprintf()` function works identically to the `fwprintf()` function. Instead of the variable list of arguments that can be passed to `fwprintf()`, `vfwprintf()` accepts its arguments in the array `arg` of type `va_list`, which must have been initialized by the `va_start()` macro (and possibly subsequent `va_arg` calls) from the `stdarg.h` header file. The `vfwprintf()` function does not invoke the `va_end` macro.

Refer to the ["wprintf" on page 676](#) function for details of the format string.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Returns the number of wide characters written or a negative number if it failed.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

["Wide Character and Byte Character Stream Orientation" on page 625](#)

["vfprintf" on page 440](#)

vswprintf

Write a formatted output to a wide character string.

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

int vswprintf(wchar_t * restrict s, size_t n,
const wchar_t * restrict format, va_list arg);
```

s	wchar_t*	The string being read
n	size_t	Number of char to print
format	wchar_t*	The format string
arg	...	Variable arguments

Remarks

The `vswprintf()` function works identically to the `swprintf()` function. Instead of the variable list of arguments that can be passed to `swprintf()`, `vswprintf()` accepts its arguments in the array `arg` of type `va_list`, which must have been initialized by the `va_start()` macro (and possibly subsequent `va_arg` calls) from the `stdarg.h` header file. The `vfwprintf()` function does not invoke the `va_end` macro.

Refer to the [“wprintf” on page 676](#) function for details of the format string.

The `vswprintf` function does not invoke the `va_end` macro.

Returns the number of characters written not counting the terminating null wide character. Otherwise a negative value if a failure occurs.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)

[“swprintf” on page 640](#)

[“vsnprintf” on page 447](#)

vwprintf

Write a formatted text to a wide character oriented stdout

```
#include <wchar.h>
int vwprintf(const wchar_t * format, va_list arg);
```

format	wchar_t*	The format string
....		Variable arguments

Remarks

The `vwprintf()` function works identically to the `wprintf()` function. Instead of the variable list of arguments that can be passed to `wprintf()`, `vwprintf()` accepts its arguments in the array `arg` of type `va_list`, which must have been initialized by the `va_start()` macro (and possibly subsequent `va_arg` calls) from the `stdarg.h` header file. The `vwprintf()` function does not invoke the `va_end` macro.

Refer to the [“wprintf” on page 676](#) function for details of the format string.

Returns the number of characters written or a negative value if it failed.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 625](#)

[“vprintf” on page 445](#)

watof

Convert a wide character string to a double type

```
#include <wchar.h>
double watof(wchar_t * str);
```

str	wchar_t	The wide character string to be converted
-----	---------	---

Remarks

Performs the same task as `atof` for a wide character type.

Returns the converted value or, if no conversion could be performed, zero.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“atof” on page 465](#)

wcrtomb

Translate a `wchar_t` type to a multibyte character according to the encoding scheme specified in the `LC_CTYPE` component of the current locale. This function is essentially the same as `wctomb()` except that it has an additional parameter of type `mbstate_t*`, which is ignored if the encoding scheme is non-modal.

```
#include <stdlib.h>
int wcrtomb(char *s, wchar_t wchar, mbstate_t * ps);
```

s	char *	A multibyte string buffer
wchar	wchar_t	A wide character to convert
ps	mbstate_t *	The current state of translation between multibyte and wide character, ignored if the encoding scheme is non-modal.

Remarks

If `s` is a null pointer, this call is equivalent to `wcrtomb(buf, L'\0', ps)` where `buf` is an internal buffer.

If `s` is not a null pointer, `wcrtomb()` determines the length of the UTF-8 multibyte string that corresponds to the wide character `wchar` and stores that string in the multibyte string pointed to by `s`. At most `MB_CUR_MAX` bytes are stored.

`wcrtomb()` returns the number of bytes stored in the string `s`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“mbrtowc” on page 635](#)

[“wcsrtombs” on page 658](#)

wcscat

Wide character string concatenation

```
#include <wchar.h>
wchar_t * wcscat(wchar_t * dst, const wchar_t * src);
```

dst	wchar_t *	The destination string
src	wchar_t *	The source string

Remarks

Performs the same task as `strcat` for a wide character type.

Returns a pointer to the destination string

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcat” on page 518](#)

wcschr

Search for an occurrence of a wide character.

```
#include <wchar.h>
wchar_t * wcschr(const wchar_t * str, const wchar_t chr);
```

str	wchar_t *	The string to be searched
chr	wchar_t	The wide character to search for

Remarks

Performs the same task as `strchr` for a wide character type.

Returns a pointer to the successfully located wide character. If it fails, `wcschr()` returns a null pointer (NULL).

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strchr” on page 519](#)

wcscmp

Compare two wide character arrays.

```
#include <wchar.h>
int wcscmp(const wchar_t * str1, const wchar_t * str2);
```

str1	wchar_t *	Comparison string
str2	wchar_t *	Comparison string

Remarks

Performs the same task as `strcmp` for a wide character type.

Returns a zero if `str1` and `str2` are equal, a negative value if `str1` is less than `str2`, and a positive value if `str1` is greater than `str2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strcmp” on page 520](#)
- [“wmemcmp” on page 673](#)

wcscoll

Compare two wide character arrays according to the collating sequence defined in the LC_COLLATE component of the current locale.

```
#include <wchar.h>
int wcscoll(const wchar_t *str1, const wchar_t * str2);
```

str1	wchar_t *	First comparison string
str2	wchar_t *	Second comparison string

Remarks

Performs the same task as `strcoll` for a wide character type.

Returns zero if `str1` is equal to `str2`, a negative value if `str1` is less than `str2`, and a positive value if `str1` is greater than `str2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strcoll” on page 522](#)
- [“wcscmp” on page 650](#)
- [“wmemcmp” on page 673](#)

wcscspn

Find the first wide character in one wide character string that is also in another.

```
#include <wchar.h>
size_t wcscspn(const wchar_t * str, const wchar_t * set);
```

str	wchar_t *	The string to be searched
set	wchar_t *	The set of characters to be searched for

Remarks

The `wcscspn()` function finds the first wide character in the `null` terminated wide character string `s1` that is also in the `null` terminated wide character string `s2`. For this purpose, the `null` terminators are considered part of the strings. The function starts examining characters at the beginning of `s1` and continues searching until a wide character in `s1` matches a wide character in `s2`.

`wcscspn()` returns the index of the first wide character in `s1` that matches a wide character in `s2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcspn” on page 525](#)

wcscpy

Copy one wide character array to another.

```
#include <wchar.h>
wchar_t * (wcscpy)(wchar_t * dst, const wchar_t * src);
```

dst	wchar_t *	The destination string
src	wchar_t *	The source being copied

Remarks

The `wcscpy()` function copies the wide character array pointed to by `src` to the wide character array pointed to `dst`. The `src` argument must point to a null terminated wide character array. The resulting wide character array at `dest` is null terminated as well.

If the arrays pointed to by `dest` and `source` overlap, the operation of `strcpy()` is undefined.

Returns a pointer to the destination string.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strcpy” on page 524](#)

wcsftime

Formats a wide character string for time.

```
#include <wchar.h>
size_t wcsftime(wchar_t * str, size_t max_size,
const wchar_t * format_str, const struct tm * timeptr);
```

<code>str</code>	<code>wchar_t *</code>	The destination string
<code>max_size</code>	<code>size_t</code>	Maximum string size
<code>format_str</code>	<code>const wchar_t *</code>	The format string
<code>timeptr</code>	<code>const struct tm *</code>	The time structure containing the calendar time

Remarks

Performs the same task as `strftime` for a wide character type.

The `wcsftime` function returns the total number of characters in the argument `str` if the total number of characters including the null character in the string argument `str` is less than the value of `max_size` argument. If it is greater, `wcsftime` returns 0

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“strftime” on page 562](#)

wcslen

Compute the length of a wide character array.

```
#include <wchar.h>
size_t (wcslen)(const wchar_t * str);
```

str	wchar_t *	The string to compute
-----	-----------	-----------------------

Remarks

The `wcslen()` function computes the number of characters in a null terminated wide character array pointed to by `str`. The null character (`L'\0'`) is not added to the wide character count.

Returns the number of characters in a wide character array not including the terminating null character.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“strlen” on page 528](#)

wcsncat

Append a specified number of characters to a wide character array.

```
#include <wchar.h>
wchar_t * wcsncat(wchar_t * dst, const wchar_t * src, size_t n);
```

dst	wchar_t *	The destination string
src	wchar_t *	The string to be appended
n	size_t	The number of characters to copy

Remarks

The `wcsncat()` function appends a maximum of `n` characters from the wide character array pointed to by `source` to the wide character array pointed to by `dest`. The `dest` argument must point to a null terminated wide character array. The `src` argument does not necessarily have to point to a null terminated wide character array.

If a null wide character is reached in `src` before `n` characters have been appended, `wcsncat()` stops.

When done, `wcsncat()` terminates `dest` with a null wide character (`L'\\0'`).

Returns a pointer to the destination string.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strncat” on page 529](#)

wcsncmp

Compare not more than a specified number of wide characters.

```
#include <wchar.h>
int csncmp(const wchar_t * str1,
           const wchar_t * str2, size_t n);
```

str1	wchar_t *	First comparison string
str2	wchar_t *	Second comparison string
n	size_t	Maximum number of characters to compare

Remarks

Performs the same task as `strncmp` for a wide character type.

Returns a zero if the first `n` characters of `str1` and `str2` are equal, a negative value if `str1` is less than `str2`, and a positive value if `str1` is greater than `str2`.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“strncmp” on page 531](#)

wcsncpy

Copy a specified number of wide characters.

```
#include <wchar.h>
wchar_t * wcsncpy(wchar_t * dst,
const wchar_t * src, size_t n);
```

dst	wchar_t *	Destination string
src	wchar_t *	Source to be copied
n	size_t	Number of characters to copy

Remarks

Performs the same task as `strncpy` for a wide character type.

Returns a pointer to the destination string.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“strncpy” on page 532](#)
[“wcscpy” on page 652](#)

wcspbrk

Look for the first occurrence of an element of an array of wide characters in another.

```
#include <wchar.h>
wchar_t * wcspbrk(const wchar_t * str, const wchar_t * set);
```

str	wchar_t*	The string being searched
set	wchar_t *	The search set

Remarks

Performs the same task as `struprbrk` for a wide character type.

Returns a pointer to the first wide character in `str` that matches any wide character in `set`, and returns a null pointer (NULL) if no match was found.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“struprbrk” on page 534](#)

wcsrchr

Search a wide character string for the last occurrence of a specified wide character.

```
#include <wchar.h>
wchar_t * wcsrchr(const wchar_t * str, wchar_t chr);
```

str	const wchar_t *	The string being searched
chr	wchar_t	The wide character to search for

Remarks

Performs the same task as `strrchr` for a wide character type.

Returns a pointer to the wide character found or returns a null pointer (NULL) if it fails.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strchr” on page 519](#)

wcsrtombs

Translate a `wchar_t` type character array to a multibyte character array. This function is essentially the same as `wcstombs()` except that it has an additional parameter of type `mbstate_t*`, which is ignored if the encoding scheme is non-modal.

```
#include <wchar.h>
size_t wcsrtombs(char *dst, const wchar_t **src, size_t n,
                 mbstate_t * ps);
```

dst	char *	The character string destination
src	const wchar_t *	Indirect pointer to the wide character string to be converted
n	size_t	The maximum length to convert
ps	mbstate_t *	The current state of translation between multibyte and wide character, ignored if the encoding scheme is non-modal.

Remarks

The MSL implementation of the `wcsrtombs()` function converts a character array containing `wchar_t` type Unicode characters indirectly pointed to by `src` to a character array containing UTF-8 multibyte characters. If `dst` is not a null pointer, these multibyte characters are stored in the array pointed to by `dst`. Conversion continues until either a terminating null wide character is

encountered or (if dst is not a null pointer) if the translation of the next wide character would cause the total number of bytes to be stored to exceed n.

If dst is not a null pointer, the wchar_t * object pointed to by src is assigned either a null pointer if conversion ended because a null wide character was reached or the address just past the last wide character converted.

wcsrtombs() returns the number of bytes modified in the character array pointed to by s, not including a terminating null character, if any.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wcrtomb” on page 648](#)

[“mbsrtowcs” on page 637](#)

wcsspn

Count the number of wide characters in one wide character array that are in another.

```
#include <wchar.h>
size_t wcsspn(const wchar_t * str, const wchar_t * set);
```

str	wchar_t *	The searched string
set	const wchar_t *	The search set

Remarks

Performs the same task as strspn for a wide character type.

Returns the number of characters in the initial segment of str that contains only characters that are elements of set.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strspn” on page 536](#)

wcsstr

Search for a wide character array within another.

```
#include <wchar.h>
wchar_t * wcsstr(const wchar_t * str, const wchar_t * pat);
```

str	const wchar_t *	The string to search
pat	const wchar_t *	The string being searched for

Remarks

Performs the same task as `strstr` for a wide character type.

Returns a pointer to the first occurrence of `s2` in `s1` and returns a null pointer (`NULL`) if `s2` cannot be found.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strstr” on page 537](#)

[“wcschr” on page 650](#)

wcstod

Converts a wide character array to double values.

```
#include <wchar.h>
double wcstod(wchar_t * str, char ** end);
```

str	wchar_t *	The string being converted
end	char **	If not null, a pointer to the first position not convertible.

Remarks

The `wcstod()` function converts a wide character array, pointed to by `nptr`, to a floating point value of type `double`. The wide character array can be in either decimal or hexadecimal floating point constant notation (e.g. `103.578`, `1.03578e+02`, or `0x1.9efef9p+6`).

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to a value of type `double`.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

Returns a floating point value of type `double`. If `str` cannot be converted to an expressible double value, `wcstod()` returns `HUGE_VAL`, defined in `math.h`, and sets `errno` to `ERANGE`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- ["wcstof" on page 661](#)
- ["strtod" on page 493](#)
- ["wcstold" on page 665](#)
- ["errno" on page 95](#)

wcstof

Wide character array conversion to floating point value of type float.

```
#include <wchar.h>
float wcstof(const wchar_t * restrict nptr,
wchart_t ** restrict endptr);
```

nptr	const wchar_t *	A Null terminated wide character array to convert
endptr	wchar_t **	A pointer to a position in nptr that follows the converted part.

Remarks

The `wcstof()` function converts a wide character array, pointed to by `nptr`, to a floating point value of type `float`. The wide character array can be in either decimal or hexadecimal floating point constant notation (e.g. `103.578`, `1.03578e+02`, or `0x1.9efef9p+6`).

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to a value of type `float`.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`wcstof()` returns a floating point value of type `float`. If `nptr` cannot be converted to an expressible float value, `wcstof()` returns `HUGE_VAL`, defined in `math.h`, and sets `errno` to `ERANGE`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“wcstod” on page 660](#)

[“wcstold” on page 665](#)

[“strtod” on page 496](#)

[“errno” on page 95](#)

wcstok

Extract tokens within a wide character array.

```
#include <wchar.h>
wchar_t * wcstok(wchar_t * str,
const wchar_t * set, wchar_t ** ptr);
```

str	wchar_t *	The string to be modified
set	wchar_t *	The list of wide character to find
ptr	wchar_t *	Continuation information

Remarks

Performs the same task as `strtok` for a wide character type however, it makes use of a third argument to contain sufficient information to continue the tokenization process.

When first called, the first argument is non-null. `wcstok()` returns a pointer to the first token in `str` or returns a null pointer if no token can be found.

Subsequent calls to `wcstok()` with a NULL `str` argument causes `wcstok()` to return a pointer to the next token or return a null pointer (NULL) when no more tokens exist. When called with a NULL `str` argument, the value of the `ptr` argument must have been set by a previous call to `wcstok()`.

The `wcstok` function returns a pointer to the first wide character of a token, or a null pointer if there is no token.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strtok” on page 539](#)

[“errno” on page 95](#)

wcstol

Wide character array conversion to floating point value of type long int.

```
#include <wchar.h>

long int wcstol( const wchar_t * restrict nptr,
    wchar_t ** restrict endptr, int base);
```

nptr	const wchar_t *	A Null terminated wide character array to convert
endptr	wchar_t **	A pointer to a position in nptr that follows the converted part
base	int	A numeric base between 2 and 36

Remarks

The `wcstol()` function converts a wide character array, pointed to by `nptr`, to an integer value of type `long`. The `base` argument in `wcstold()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters `a` (or `A`) through `z` (or `Z`) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then `wcstold()` converts the wide character array based on its format. Wide character arrays beginning with '`0`' are assumed to be octal, number strings beginning with '`0x`' or '`0X`' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to a value of type `long int`.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`wcstol()` returns an signed integer value of type `long int`. If the converted value is less than `LONG_MIN`, `wcstol()` returns `LONG_MIN` and sets `errno` to `ERANGE`. If the converted value is greater than `LONG_MAX`, `wcstol()` returns `LONG_MAX` and sets `errno` to `ERANGE`. The `LONG_MIN` and `LONG_MAX` macros are defined in `limits.h`

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“strtol” on page 497](#)
- [“errno” on page 95](#)
- [“wcstoll” on page 666](#)
- [“wcstoul” on page 667](#)

wcstold

A wide character array conversion to an floating point value of type long double.

```
#include <wchar.h>

long double wcstold(const wchar_t * restrict nptr,
                     wchar_t ** restrict endptr);
```

nptr	const wchar_t *	A Null terminated wide character array to convert
endptr	wchar_t **	A pointer to a position in nptry that is not convertible.

Remarks

The `wcstold()` function converts a wide character array, pointed to by `nptr`, expected to represent an integer expressed in radix base, to an integer value of type `long int`. A plus or minus sign (+ or -) prefixing the number string is optional. The wide character array can be in either decimal or hexadecimal floating point constant notation (e.g. 103.578, 1.03578e+02, or 0x1.9efef9p+6).

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to a double value.

In other than the “C” locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

Returns a floating point value of type `long double`. If `nptr` cannot be converted to an expressible double value, `wcstold()` returns `HUGE_VAL`, defined in `math.h`, and sets `errno` to `ERANGE`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wcstod” on page 660](#)

[“wcstof” on page 661](#)

[“strtold” on page 500](#)

wcstoll

Wide character array conversion to integer value of type `long long int`.

```
#include <wchar.h>
long long int wcstoll(const wchar_t * restrict nptr,
                      wchar_t ** restrict endptr, int base);
```

<code>nptr</code>	<code>const wchar_t *</code>	A Null terminated wide wide character array to convert
<code>endptr</code>	<code>wchar_t **</code>	A pointer to a position in <code>nptr</code> that is not convertible.
<code>base</code>	<code>int</code>	A numeric base between 2 and 36

Remarks

The `wcstoll()` function converts a wide character array, pointed to by `nptr`, expected to represent an integer expressed in radix `base` to an integer value of type `long long int`. A plus or minus sign (+ or -) prefixing the number string is optional.

The `base` argument in `wcstoll()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then `wcstoll()` converts the wide character array based on its format. Wide character arrays beginning with '0' are

assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to a long long int value.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`wcstoll()` returns an integer value of type long long int. If the converted value is less than `LLONG_MIN`, `wcstoll()` returns `LLONG_MIN` and sets `errno` to `ERANGE`. If the converted value is greater than `LLONG_MAX`, `wcstoll()` returns `LLONG_MAX` and sets `errno` to `ERANGE`. The `LLONG_MIN` and `LLONG_MAX` macros are defined in `limits.h`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- ["strtoll" on page 501](#)
- ["wcstol" on page 664](#)
- ["wcstoul" on page 667](#)
- ["errno" on page 95](#)

wcstoul

Wide character array conversion to integer value of type unsigned long int.

```
#include <wchar.h>
unsigned long int wcstoul(const wchar_t * restrict nptr,
                           wchar_t ** restrict endptr, int base);
```

<code>nptr</code>	<code>const wchar_t *</code>	A Null terminated wide character array to convert
-------------------	------------------------------	---

endptr	wchar_t **	A pointer to a position in nptr that is not convertible.
base	int	A numeric base between 2 and 36

Remarks

The `wcstoul()` function converts a wide character array, pointed to by `nptr`, to an integer value of type `unsigned long int`, in `base`. A plus or minus sign prefix is ignored.

The `base` argument in `wcstoul()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then `strtol()` and `wcstoul()` convert the wide character array based on its format. Wide character arrays beginning with '0' are assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to the functions' respective types.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`wcstoul()` returns an unsigned integer value of type `unsigned long int`. If the converted value is greater than `ULONG_MAX`, `wcstoul()` returns `ULONG_MAX` and sets `errno` to `ERANGE`. The `ULONG_MAX` macro is defined in `limits.h`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

["wcstol" on page 664](#)

["wcstoll" on page 666](#)

["wcstoull" on page 669](#)

["strtoul" on page 502](#)

["errno" on page 95](#)

wcstoull

Wide character array conversion to integer value of type unsigned long long int.

```
#include <wchar.h>
unsigned long long int wcstoull(
    const wchar_t * restrict nptr,
    wchar_t ** restrict endptr, int base);
```

nptr	const wchar_t *	A Null terminated wide character array to convert
endptr	wchar_t **	A pointer to a position in nptr that is not convertible.
base	int	A numeric base between 2 and 36

Remarks

The `wcstoull()` function converts a wide character array, pointed to by `nptr`, expected to represent an integer expressed in radix `base` to an integer value of type `unsigned long long int`. A plus or minus sign prefix is ignored.

The `base` argument in `wcstoull()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. The letters a (or A) through z (or Z) are used for the values 10 through 35; only letters and digits representing values less than `base` are permitted. If `base` is 0, then `wcstoull()` converts the wide character array based on its format. Wide character arrays beginning with '0' are assumed to be octal, number strings beginning with '0x' or '0X' are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a `null` pointer, it is assigned a pointer to a position within the wide character array pointed to by `nptr`. This position marks the first wide character that is not convertible to a `long long` value.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

`wcstoull()` returns an unsigned integer value of type `unsigned long long int`. If the converted value is greater than `ULLONG_MAX`,

wcstoull() returns ULONG_MAX and sets errno to ERANGE. The ULONG_MAX macro is defined in limits.h

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wcstol” on page 664](#)

[“wcstoll” on page 666](#)

[“wcstoul” on page 667](#)

[“strtoul” on page 502](#)

[“errno” on page 95](#)

wcsxfrm

Transform a wide character array as specified in the LC_COLL component of the current locale.

```
#include <wchar.h>
size_t wcsxfrm(wchar_t * str1, const wchar_t * str2, size_t n);
```

str1	wchar_t *	The destination string
str2	wchar_t *	The source string
n	size_t	Maximum number of characters

Remarks

Performs the same task as `strxfrm` for a wide character type.

Returns the length of the transformed wide string in `str1`, not including the terminating null wide character. If the value returned is `n` or greater, the contents of `str1` are indeterminate.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“strxfrm” on page 540](#)

wctime

Convert a time_t type to a wide character array

```
#include <wchar.h>
wchar_t * wctime(const time_t * timer);
```

timer	const time_t *	The Calendar Time
-------	----------------	-------------------

Remarks

Performs the same task as `ctime` for a wide character type.

Returns a pointer to wide character array containing the converted `time_t` type

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ctime” on page 555](#)

wctob

The function `wctob()` converts a wide character to a byte character.

```
#include <wchar.h>
int wctob(wint_t wc);
```

int	wchar_t *	The wide character to be converted
-----	-----------	------------------------------------

Returns

The function `wctob()` returns the single byte representation of the argument `wc` as an unsigned char converted to an int or EOF is returned if `wc` does not correspond to a valid multibyte character.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“wcrtomb” on page 648](#)

wmemchr

Search for an occurrence of a specific wide character.

```
#include <wchar.h>
void * wmemchr(const void * src, int val, size_t n);
```

src	const void *	The string to be searched
val	int	The value to search for
n	size_t	The maximum length of a search

Remarks

Performs the same task as `memchr()` for a wide character type.

Returns a pointer to the found wide character, or a null pointer (NULL) if `val` cannot be found.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“memchr” on page 512](#)
[“wcschr” on page 650](#)

wmemcmp

Compare two blocks of memory, treated as wide characters.

```
#include <wchar.h>
int wmemcmp(const void * src1, const void * src2, size_t n);
```

src1	const void *	First memory block to compare
src2	const void *	Second memory block to compare
n	size_t	Maximum number of wide characters to compare

Remarks

Performs the same task as `memcmp()` for a wide character type.

The function `wmemcmp` returns a zero if all `n` characters pointed to by `src1` and `src2` are equal.

The function `wmemcmp` returns a negative value if the first non-matching wide character pointed to by `src1` is less than the wide character pointed to by `src2`.

The function `wmemcmp` returns a positive value if the first non-matching wide character pointed to by `src1` is greater than the wide character pointed to by `src2`.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memcmp” on page 515](#)

[“wcscmp” on page 650](#)

wmemcpy

Copy a contiguous memory block.

```
#include <wchar.h>
void * (wmemcpy)(void * dst, const void * src, size_t n);
```

dst	void *	The destination string
src	const void *	The source string
n	size_t	Maximum length to copy

Remarks

Performs the same task as `memcpy()` for a wide character type.

Returns a pointer to the destination string.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memcpy” on page 516](#)

wmemmove

Copy an overlapping contiguous memory block.

```
#include <wchar.h>
void * (wmemmove)(void * dst, const void * src, size_t n);
```

dst	void *	The destination string
src	const void *	The source string
n	size_t	The maximum length to copy

Remarks

Performs the same task as `memmove()` for a wide character type.

Returns a pointer to the destination string.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memmove” on page 517](#)

[“wcscpy” on page 652](#)

wmemset

Clear the contents of a block of memory.

```
#include <wchar.h>
void * wmemset(void * dst, int val, size_t n);
```

dst	void *	The destination string
val	int	The value to be set
n	size_t	The maximum length

Remarks

Performs the same task as `memset()` for a wide character type.

Returns a pointer to the destination string

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“memset” on page 518](#)

wprintf

Send formatted wide character text to a standard output.

```
#include <wchar.h>
int wprintf(const wchar_t * format, ...);
```

format	wchar_t*	The format string
....		Variable arguments

Remarks

Performs the same task as `printf()` for a wide character type.

Returns the number of arguments written or a negative number if an error occurs.

Listing 44.4 Length Modifiers And Conversion Specifiers For Formatted Output Functions

Modifier	Description
Size	
h	The h flag followed by d, i, o, u, x, or X conversion specifier indicates that the corresponding argument is a short int or unsigned short int.
l	The lower case L followed by d, i, o, u, x, or X conversion specifier indicates the argument is a long int or unsigned long int. The lower case L followed by a c conversion specifier, it indicates that the argument is of type <code>wint_t</code> . The lower case L followed by an s conversion specifier, it indicates that the argument is of type <code>wchar_t</code> .
ll	The double l followed by d, i, o, u, x, or X conversion specifier indicates the argument is a long long or unsigned long long
L	The upper case L followed by e, E, f, g, or G conversion specifier indicates a long double.

Listing 44.4 Length Modifiers And Conversion Specifiers For Formatted Output Functions

v	AltiVec: A vector bool char, vector signed char or vector unsigned char when followed by c, d, i, o, u, x or X A vector float, when followed by f.
vh hv	AltiVec: A vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X
vl lv	AltiVec: A vector int, vector unsigned int or vector bool int when followed by c, d, i, o, u, x or X

Flags

-	The conversion will be left justified.
+	The conversion, if numeric, will be prefixed with a sign (+ or -). By default, only negative numeric values are prefixed with a minus sign (-).
space	If the first character of the conversion is not a sign character, it is prefixed with a space. Because the plus sign flag character (+) always prefixes a numeric value with a sign, the space flag has no effect when combined with the plus flag.
#	For c, d, i, and u conversion types, the # flag has no effect. For s conversion types, a pointer to a Pascal string, is output as a character string. For o conversion types, the # flag prefixes the conversion with a 0. For x conversion types with this flag, the conversion is prefixed with a 0x. For e, E, f, g, and G conversions, the # flag forces a decimal point in the output. For g and G conversions with this flag, trailing zeroes after the decimal point are not removed.
0	This flag pads zeroes on the left of the conversion. It applies to d, i, o, u, x, X, e, E, f, g, and G conversion types. The leading zeroes follow sign and base indication characters, replacing what would normally be space characters. The minus sign flag character overrides the 0 flag character. The 0 flag is ignored when used with a precision width for d, i, o, u, x, and X conversion types.
@	AltiVec This flag indicates a pointer to a string specified by an argument. This string will be used as a separator for vector elements.

Conversions

Listing 44.4 Length Modifiers And Conversion Specifiers For Formatted Output Functions

d	The corresponding argument is converted to a signed decimal.
i	The corresponding argument is converted to a signed decimal.
o	The argument is converted to an unsigned octal.
u	The argument is converted to an unsigned decimal.
x, X	The argument is converted to an unsigned hexadecimal. The x conversion type uses lowercase letters (abcdef) while X uses uppercase letters (ABCDEF).
n	This conversion type stores the number of items output by printf() so far. Its corresponding argument must be a pointer to an int.
f, F	The corresponding floating point argument (float, or double) is printed in decimal notation. The default precision is 6 (6 digits after the decimal point). If the precision width is explicitly 0, the decimal point is not printed. For the f conversion specifier, a double argument representing infinity produces [-]inf; a double argument representing a NaN (Not a number) produces [-]nan. For the F conversion specifier, [-]INF or [-]NAN are produced instead.
e, E	The floating point argument (float or double) is output in scientific notation: [-]b.aaa±Eee. There is one digit (b) before the decimal point. Unless indicated by an optional precision width, the default is 6 digits after the decimal point (aaa). If the precision width is 0, no decimal point is output. The exponent (ee) is at least 2 digits long. The e conversion type uses lowercase e as the exponent prefix. The E conversion type uses uppercase E as the exponent prefix.
g, G	The g conversion type uses the f or e conversion types and the G conversion type uses the f or E conversion types. Conversion type e (or E) is used only if the converted exponent is less than -4 or greater than the precision width. The precision width indicates the number of significant digits. No decimal point is output if there are no digits following it.
c	The corresponding argument is output as a character.

Listing 44.4 Length Modifiers And Conversion Specifiers For Formatted Output Functions

s	If the <code>s</code> format specifier is preceded by an <code>l</code> length modifier, the argument shall be a pointer to the initial element of an array of <code>wchar_t</code> type that is null terminated If there is no <code>l</code> length modifier present, the argument shall be a pointer to the initial element of a character array containing a multibyte character sequence beginning in the initial shift state..
p	The corresponding argument is taken to be a pointer. The argument is output using the <code>x</code> conversion type format.
CodeWarrior Extensions	
#s	The corresponding argument, a pointer to a Pascal string, is output as a character string. A Pascal character string is a length byte followed by the number characters specified in the length byte. Note: This conversion type is an extension to the ANSI C library but applied in the same manner as for other format variations.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 625](#)
- [“printf” on page 402](#)
- [“fwprintf” on page 631](#)

wscanf

Reads a wide character formatted text from standard input

```
#include <wchar.h>
int wscanf(const wchar_t * format, ...);
```

format	wchar_t*	The format string
....		Variable arguments

Remarks

Performs the same task as `scanf()` for a wide character type.

Returns the number of items successfully read and returns `WEOF` if a conversion type does not match its argument or an end-of-file is reached.

Listing 44.5 Length Modifiers And Conversion Specifiers For Input Functions

Modifier	Description of Length Specifiers
hh	The hh flag indicates that the following d, i, o, u, x, X or n conversion specifier applies to an argument that is of type char or unsigned char.
h	The h flag indicates that the following d, i, o, u, x, X or n conversion specifier applies to an argument that is of type short int or unsigned short int.
l	When used with integer conversion specifier, the l flag indicates long int or an unsigned long int type. When used with floating point conversion specifier, the l flag indicates a double. When used with a c or s conversion specifier, the l flag indicates that the corresponding argument with type pointer to wchar_t.
ll	When used with integer conversion specifier, the ll flag indicates that the corresponding argument is of type long long or an unsigned long long.
L	The L flag indicates that the corresponding float conversion specifier corresponds to an argument of type long double.
v	AltiVec: A vector bool char, vector signed char or vector unsigned char when followed by c, d, i, o, u, x or X A vector float, when followed by f.
vh hv	AltiVec: vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X
vl lv	AltiVec: vector long, vector unsigned long or vector bool when followed by c, d, i, o, u, x or X
Conversion Specifiers	
d	A decimal integer is read.

Listing 44.5 Length Modifiers And Conversion Specifiers For Input Functions

i	A decimal, octal, or hexadecimal integer is read. The integer can be prefixed with a plus or minus sign (+, -), 0 for octal numbers, 0x or 0X for hexadecimal numbers.
o	An octal integer is read.
u	An unsigned decimal integer is read.
x, X	A hexadecimal integer is read.
e, E, f, g, G	A floating point number is read. The number can be in plain decimal format (e.g. 3456.483) or in scientific notation ([-] b. aaaa \pm dd).
s	If the format specifier s is preceded with an l (el) length modifier, then the corresponding argument must be a pointer to an array of wchar_t. This array must be large enough to accept the sequence of wide_characters being read, including the terminating null character, automatically appended. If there is no preceding l length modifier then the corresponding argument must be an array of characters. The wide-characters read from the input field will be converted to a sequence of multibyte characters before being assigned to this array. That array must be large enough to accept the sequence of multibyte characters including the terminating null character automatically appended.
c	A character is read. White space characters are not skipped, but read using this conversion specifier..
p	A pointer address is read. The input format should be the same as that output by the p conversion type in printf().
n	This conversion type does not read from the input stream but stores the number of characters read in its corresponding argument.
[scanfset]	Input stream characters are read and filtered determined by the scanfset. See “wscanf” on page 679 , for a full description.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 625](#)
- [“scanf” on page 420](#)
- [“fwscanf” on page 632](#)

Non Standard <wchar.h> Functions

Various non standard functions are included in the header `wchar.h` for legacy source code and compatibility with operating system frameworks and application programming interfaces.

- For the function `wcsdup` see [“wcsdup” on page 128](#), for a full description.
- For the function `wcsicmp` see [“wcsicmp” on page 129](#), for a full description.
- For the function `wcslwr` see [“wcslwr” on page 130](#), for a full description.
- For the function `wcsncmp` see [“wcsncmp” on page 132](#), for a full description.
- For the function `wcsnset` see [“wcsnset” on page 133](#), for a full description.
- For the function `wcsrev` see [“wcsrev” on page 134](#), for a full description.
- For the function `wcsset` see [“wcsset” on page 134](#), for a full description.
- For the function `wcsspnp` see [“wcsspnp” on page 135](#), for a full description.
- For the function `wcsupr` see [“wcsupr” on page 135](#), for a full description.
- For the function `wtoi` see [“wtoi” on page 136](#), for a full description.

wctype.h

The `ctype.h` header file supplies macros and functions for testing and manipulation of wide character type.

Overview of wctype.h

The header `wctype.h` has several testing and conversion functions and types.

- [“wctype.h types” on page 684](#) defines two types used for wide character values
- [“iswalnum” on page 684](#) tests for alpha-numeric wide characters
- [“iswalpha” on page 685](#) tests for alphabetical wide characters
- [“iswblank” on page 685](#) tests for a blank space or space holder
- [“iswcntrl” on page 686](#) tests for control wide characters
- [“iswdigit” on page 686](#) tests for digital wide characters
- [“iswgraph” on page 687](#) tests for graphical wide characters
- [“iswlower” on page 688](#) tests for lower wide characters
- [“iswprint” on page 688](#) tests for printable wide characters
- [“iswpunct” on page 689](#) tests for punctuation wide characters
- [“iswspace” on page 689](#) tests for whitespace wide characters
- [“iswupper” on page 690](#) tests for uppercase wide characters
- [“iswdx digit” on page 691](#) tests for a hexadecimal wide character type
- [“towlower” on page 692](#) converts wide characters to lower case
- [“towupper” on page 692](#) converts wide characters to upper case

Mapping facilities

- [“towctrans” on page 691](#) a case and wide character mapping function
- [“wctrans” on page 693](#) a wide character mapping function

Types

The header `wctype.h` contains two types described in the table “[wctype.h types](#)” on [page 684](#) used for wide character manipulations.

Listing 45.1 wctype.h types

wctrans_t	A scalar type which can represent locale specific character mappings
wctype_t	A scalar type that can represent locale specific character classifications

iswalnum

Tests for alpha-numeric wide characters.

```
#include <wctype.h>
int iswalnum (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as `isalnum` for wide character type.

In the “C” locale `true` is returned for an alphanumeric: [a-z], [A-Z], [0-9]

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“`isalnum`” on page 68](#)

iswalpha

Tests for alphabetical wide characters.

```
#include <wctype.h>
int iswalpha (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as isalpha for wide character type.

In the “C” locale true is returned for an alphabetic: [a-z], [A-Z]

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isalpha” on page 71](#)

iswblank

Tests for a blank space or a word separator dependent upon the locale usage.

```
#include <wctype.h>
int isblank(win_t c);
```

c	win_t	character being evaluated
---	-------	---------------------------

Remarks

This function determines if a wide character is a blank space or tab or if the wide character is in a locale specific set of wide characters for which iswspace is true and is used to separate words in text.

In the “C” locale, isblank returns true only for the space and tab characters.

Return

True is returned if the criteria are met.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“iswspace” on page 689](#)

iswcntrl

Tests for control wide characters.

```
#include <wctype.h>
int iswcntrl (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as iscntrl for wide character type.

True for the delete character (0x7F) or an ordinary control character from 0x00 to 0x1F.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSL/Compatibility> for a Compatibility list.

See Also

[“iscntrl” on page 72](#)

iswdigit

Tests for digital wide characters.

```
#include <wctype.h>
int iswdigit (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as isdigit for wide character type.

In the “C” locale `true` is returned for a numeric character: [0–9].

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isdigit” on page 73](#)

iswgraph

Tests for graphical wide characters.

```
#include <wctype.h>  
  
int iswgraph (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as isgraph for wide character type.

In the “C” locale `true` is returned for a non-space printing character from the exclamation (0x21) to the tilde (0x7E).

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isgraph” on page 74](#)

iswlower

Tests for lowercase wide characters.

```
#include <wctype.h>
int iswlower (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as islower for wide character type.

In the “C” locale `true` is returned for a lowercase letter: [a–z].

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“islower” on page 74](#)

[“iswupper” on page 690](#)

iswprint

Tests for printable wide characters.

```
#include <wctype.h>
int iswprint (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as isprint for wide character type.

In the “C” locale `true` is returned for a printable character from space (0x20) to tilde (0x7E).

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isprint” on page 75](#)

iswpunct

Tests for punctuation wide characters.

```
#include <wctype.h>
int iswpunct (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as ispunct for wide character type.

True for a punctuation character. A punctuation character is neither a control nor an alphanumeric character.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“ispunct” on page 76](#)

iswspace

Tests for whitespace wide characters.

```
#include <wctype.h>
int iswspace (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as isspace for wide character type.

In the “C” locale `true` is returned for a space, tab, return, new line, vertical tab, or form feed.

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“isspace” on page 76](#)

iswupper

Tests for uppercase wide characters.

```
#include <wctype.h>
int iswupper (wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as isupper for wide character type.

In the “C” locale `true` is returned for an uppercase letter: [A-Z].

This function may not be implemented on all platforms. Please refer to [<http://www.metrowerks.com/docs/MSLCompatibility>](http://www.metrowerks.com/docs/MSLCompatibility) for a Compatibility list.

See Also

[“isupper” on page 77](#)

[“iswlower” on page 688](#)

iswxdigit

Tests for a hexadecimal wide character type.

```
#include <wctype.h>
int iswxdigit(wchar_t wc);
```

wc	wchar_t	The wide character to test
----	---------	----------------------------

Remarks

Provides the same functionality as isxdigit for wide character type.

True for a hexadecimal digit [0-9], [A-F], or [a-f].

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“isxdigit” on page 78](#)

towctrans

Maps a wide character type to another wide character type.

```
#include <wchar.h>
wint_t towctrans(wint_t c, wctrans_t value);
```

c	wint_t	The character to remap
value	wctrans_t	A value retuned by wctrans

Remarks

Maps the first argument to an upper or lower value as specified by value.

Returns the remapped character.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“wctrans” on page 693](#)

towlower

Converts wide characters from upper to lowercase.

```
#include <wctype.h>
wchar_t towlower (wchar_t wc);
```

wc	wchar_t	The wide character to convert
----	---------	-------------------------------

Remarks

Provides the same functionality as tolower for wide character type.

The lowercase equivalent of a uppercase letter and returns all other characters unchanged

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“tolower” on page 78](#)

[“towupper” on page 692](#)

towupper

Converts wide characters from lower to uppercase.

```
#include <wctype.h>
wchar_t towupper (wchar_t wc);
```

wc	wchar_t	The wide character to convert
----	---------	-------------------------------

Remarks

Provides the same functionality as toupper for wide character type.

The uppercase equivalent of a lowercase letter and returns all other characters unchanged.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“toupper” on page 79](#)

[“towlower” on page 692](#)

wctrans

Constructs a property value for “toupper” and “tolower” for character remapping.

```
#include <wchar.h>
wctrans_t wctrans(const char *name);
```

name	const char *	toupper or tolower property
------	--------------	-----------------------------

Remarks

Constructs a value that represents a mapping between wide characters. The value of name can be either toupper or tolower.

A wctrans_t type

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“towctrans” on page 691](#)

wctype.h

Overview of wctype.h

WinSIOUX.h

The SIOUX and WinSIOUX (Simple Input and Output User eXchange) libraries handle Graphical User Interface issues. Such items as menus, windows, and events are handled so your program doesn't need to for C, Pascal and C++ programs.

Overview of WinSIOUX

The following section describes the Windows versions of the console emulation interface known as WinSIOUX. The facilities and structure members for the Windows Standard Input Output User eXchange console interface are

- [“Using WinSIOUX” on page 695](#) A description of SIOUX properties.
- [“WinSIOUX for Windows” on page 696](#) the (Simple Input and Output User eXchange) library for Windows 95 and Windows NT

NOTE If you're porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

[“MSL Extras Library Headers” on page 30](#), for information on POSIX naming conventions.

Using WinSIOUX

Sometimes you need to port a program that was originally written for a command line interface such as DOS or UNIX. Or you need to write a new program quickly and don't have the time to write a complete Graphical User Interface that handles windows, menus, and events.

To help you, Metrowerks provides you with the WinSIOUX libraries, which handles all the Graphical User Interface items such as menus, windows, and titles so your program doesn't need to. It creates a window that's much like a dumb terminal or TTY

but with scrolling. You can write to it and read from it with the standard C functions and C++ operators, such as printf(), scanf(), getchar(), putchar() and the C++ inserter and extractor operators << and >>. The SIOUX and WinSIOUX libraries also creates a File menu that lets you save and print the contents of the window. There is also an Edit menu that lets you cut, copy, and paste the contents in the window.

This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

See Also

[“Overview of console.h” on page 55.](#)

NOTE If you’re porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

WinSIOUX for Windows

The WinSIOUX window is a re-sizeable, scrolling text window, where your program reads and writes text.

With the commands from the Edit menu, you can cut and copy text from the WinSIOUX window and paste text from other applications into the WinSIOUX window. With the commands in the File menu, you can print or save the contents of the WinSIOUX window

- [“Creating a Project with WinSIOUX” on page 696](#) basic steps to create a WinSIOUX program.
- [“Customizing WinSIOUX” on page 697](#) settings used to create the WinSIOUX console
- [“clrscr” on page 698](#) is used to clear the WinSIOUX console screen and buffer

Creating a Project with WinSIOUX

To use the WinSIOUX library, create a project from a project stationery that creates a WinSIOUX Console style project.

A Win SIOUX project must contain at least these libraries:

- ANSIC_WINSIOUX.LIB
- ANSIC_WINSIOUXD.LIB

- Mwcrt.lib
- MWCRTD.lib

The Win32SDK libraries:

- Winspool.lib
- Comdlg32.lib
- Gdi32.lib
- Kernel32.lib
- User32.lib

And the resource file:

WinSIOUX.rc

Customizing WinSIOUX

WinSIOUX offers the user a limited ability to customize the WinSIOUX window display. The following sections describe how you achieve this customization by modifying the structure SIOUXSettings, of type tSIOUXSettings. WinSIOUX examines some of the data fields of SIOUXSettings to determine how to create the WinSIOUX window and environment.

NOTE	To customize WinSIOUX, you must modify SIOUXSettings before you call any function that uses standard input or output. If you modify SIOUXSettings afterwards, WinSIOUX does not change its window.
-------------	--

Listing 46.1 The SIOUXSettings structure

This field...	Specifies...
char initializeTB	Not applicable to WinSIOUX.
char standalone	Not applicable to WinSIOUX.
char setupmenus	Not applicable to WinSIOUX.

Listing 46.1 The SIOUXSettings structure

This field...	Specifies...
char autocloseonquit	Whether to close the window and quit the application automatically when the program has completed.
char asktosaveonclose	Query the user whether to save the WinSIOUX output as a file, when the program is done.
char showstatusline	Not applicable to WinSIOUX.
short tabspaces	If greater than zero, substitute a tab with that number of spaces. If zero, print the tabs.
short column	The number of characters per line that the SIOUX window will contain.
short rows	The number of lines of text that the SIOUX window will contain.
short toppixel	Not applicable to WinSIOUX.
short leftpixel	Not applicable to WinSIOUX.
short fontname[32]	The font to be used in the WinSIOUX window.
short fontsize	The size of the font to be used in the WinSIOUX window.
short fontface	Not applicable to WinSIOUX.

clrscr

Clears the WinSIOUX window and flushes the buffers.

```
#include <WinSIOUX.h>
void clrscr(void);
```

Remarks

This function simply calls the function WinSIOUXclrscr, that clears the screen.

This function is not implemented directly on Windows console.

Windows compatible, only.

MSL Flags

This appendix contains a description of the macros and defines that are used as switches or flags in the MSL C library.

Overview of the MSL Switches, Flags and Defines

The MSL C library has various flags that may be set to customize the library to users specifications these include:

- “ [ANSI OVERLOAD](#) ” on page 702
- “ [MSL C LOCALE ONLY](#) ” on page 702
- “ [MSL_IMP_EXP](#) ” on page 702
- “ [MSL_INTEGRAL_MATH](#) ” on page 703
- “ [MSL_MALLOC_0 RETURNS_NON_NULL](#) ” on page 703
- “ [MSL_NEEDS_EXTRAS](#) ” on page 703
- “ [MSL_OS_DIRECT_MALLOC](#) ” on page 704
- “ [MSL_CLASSIC_MALLOC](#) ” on page 704
- “ [MSL_USE_NEW_FILE_APIS](#) ” on page 704
- “ [MSL_USE_OLD_FILE_APIS](#) ” on page 705
- “ [MSL_POSIX](#) ” on page 705
- “ [MSL_STRError_Knows_Error_Names](#) ” on page 705
- “ [SET_ERRNO](#) ” on page 706

MSL Flags

Overview of the MSL Switches, Flags and Defines

ANSI_OVERLOAD

When defined (and when using the C++ compiler) the math functions are overloaded with float and long double versions as specified by 26.5 paragraph 6 of the C++ standard. Disabling this flag removed the overloaded functions.

MSL_C_LOCALE_ONLY

The flag _MSL_C_LOCALE_ONLY provides for disabling the locale mechanism in the library.

The MSL C library should be recompiled after changing the value of the flag.

When off, the locale mechanism works as described by the ANSI C standard. When on, there are no locales, and anything which is dependent upon locales will function as if the "C" locale is in place. This saves code size and increases execution speed slightly.

MSL_IMP_EXP

This macro determines how the standard headers are decorated for importing and exporting symbols from/to a shared version of the standard runtime/C/C++ library.

If this macro is defined to nothing then the headers are configured for linking against static libraries.

In the header file `UsedDLLPrefix.h` the macro is defined to `__declspec(dllimport)`. This will allow you to link against one of the supplied shared libraries. Other related macros

```
_MSL_IMP_EXP_C  
_MSL_IMP_EXP_SIOUX  
_MSL_IMP_EXP_RUNTIME
```

allow you to link with “part” of the shared library and an individual static library.

By default the previous macros are set to whatever _MSLIMP_EXP is set to (see ansi_parms.h). As an example, if you wish to link in the static version of the SIOUX library then you would edit `ansi_parms.h` and define `_MSL_IMP_EXP_SIOUX` to nothing and link in the appropriate static `SIOUX.lib`.

_MSL_INTEGRAL_MATH

When defined (and when `__ANSI_OVERLOAD__` is defined and when using the C++ compiler), additional overloads to the math functions with integral arguments are created. This flag is meant to prevent compile time errors for statements like:

```
double c = cos(0);  
// ambiguous with __ANSI_OVERLOAD__  
// and not _MSL_INTEGRAL_MATH
```

_MSL_MALLOC_0 RETURNS_NON_NULL

This flag determines the implementation of a null allocated malloc statement. If not defined `malloc(0)` returns 0. If defined `malloc(0)` returns non-zero.

The C lib must be recompiled when flipping this switch.

_MSL_NEEDS_EXTRAS

Macintosh and Windows programs have the ability to use extra C functions when they have the MSL Extras Library linked into their project. This flag determines if they can be accessed from a C standard header or not. The flag `_MSL_NEEDS_EXTRAS` default setting is `true` for Windows and `false` for the Macintosh.

Macintosh developers must specify a non standard header to access the non standard extra functions. If Macintosh programmers wishes to access non

MSL Flags

Overview of the MSL Switches, Flags and Defines

standard functions via a standard header in legacy code, they need to do is to set `_MSL_NEEDS_EXTRAS` to `true` in `ansi_prefix.mac.h`

Windows users can access the functions in a standard header to be legacy compatible as well as the actual header where they are declared. If a Windows programmer does not want to access non standard functions via a standard header then they need to turn off `_MSL_NEEDS_EXTRAS` in `ansi_prefix.Win32.h`.

_MSL_OS_DIRECT_MALLOC

If defined `malloc` will call straight through to the OS without forming memory pools. This will likely degrade performance but may be valuable for debugging purposes.

The C lib must be recompiled when flipping this switch.

_MSL_CLASSIC_MALLOC

Enables the version of malloc that was in Pro 4 (for backward compatibility).

The C lib must be recompiled when flipping this switch.

_MSL_USE_NEW_FILE_APIS

You can use this flags to turn on and off the new-style use of the new HFS+ file system APIs, present starting with Mac OS 9.0. Using the new file system APIs gets filenames greater than 32 characters and file sizes of greater than 2GB. However, due to the standard interfaces for C, you can only access files in 2GB chunks.

Prototypes for `fread()`, for example, take a `long` as parameter for the amount of data to be read from a file, so to access more than 2GB in a file, you must make successive calls to `fread()`.

The C lib must be recompiled when flipping this switch.

_MSL_USE_OLD_FILE_APIS

You can use this flag to turn on and off the old-style use of the Mac file system APIs. Using the old file system APIs limits you to filenames of 32 characters or less and file sizes of 2GB or less.

The C lib must be recompiled when flipping this switch.

_MSL_POSIX

This flag adds the `POSIX` function `stat` to the global namespace. This is on by default.

Turning this off should allow the user to link against a third party `POSIX` library with the same names. The `POSIX` names preceded by a leading underscore are always available in MSL. For example `_open` and `_stat`.

Remarks

This macro is located in the prefix files `ansi_prefix.win32.h` or `ansi_prefix.macos.h`.

Commenting this out does not require recompilation of the library.

_MSL_STRERROR_KNOWS_ERROR_NAMES

The flag `_MSL_STRERROR_KNOWS_ERROR_NAMES` controls what happens when the `strerror()` call is made.

When the flag is on, `strerror()` will return robust error messages. When the flag is off, `strerror()` will always return with an "unknown" error. This provides for a huge savings in code/data size in the executable.

SET_ERRNO

If this flag is defined it will cause the standard math functions to set the global `errno` to either `EDOM` or `ERANGE` as specified in the 1989 C standard. The modern C standard specifies that setting `errno` for the mathlib is optional. So in the spirit of the new standard on x86 it is now optional.

Turning this off will improve performance by reducing checks for incorrect input values but will not set `errno`. Most of the standard math functions are now in the header `math_x87.h`.

Remarks

Since these definitions are in a header file they can be configured when building your application and therefore you do NOT need to rebuild the library. This switch is on by default.

This flag is only for Win32 x86 libraries.

Index

Symbols

_ANSI_OVERLOAD_ 702
_path2fss 301
_SET_ERRNO_ 706
_ttyname 60
_beginthread 275
_beginthreadex 276
_chdrive 101
_chsize 102
_clrscr 44
_creat 140
_CRTStartup 65
_DllTerminate 64
_dup 584
_dup2 585
_endthread 277
_endthreadex 278
_Exit 479
_fcntl 141
_fcreator 607
_filelength 102
_fileno 361
_findclose 182
_finddata_t 181
_findfirst 183
_findnext 183
_ftune 572
_ftype 608
_fullpath 104
_gcvt 105
_get_osfhandle 106
_getch 44
_getche 45
_getdcwd 81
_getdiskfree 82
_getdrive 105
_getdrives 82
_gotoxy 45
_HandleTable 65
_heapmin 107
_initscr 46
_inp 46
_inp4 47
_inpw 47
_IOFBF 428
_IOLBF 428
_IONBF 428
_itoa 107
_itow 108
_kbhit 48
_ltoa 109
_ltow 109
_makepath 110
_MSL_C_LOCALE_ONLY 702
_MSL_CLASSIC_MALLOC 704
_MSL_IMP_EXP 702
_MSL_IMP_EXP_C 702
_MSL_IMP_EXP_RUNTIME 702
_MSL_IMP_EXP_SIOUX 702
_MSL_INTEGRAL_MATH 703
_MSL_MALLOC_0_RETURNS_NON_NULL 703
_MSL_NEEDS_EXTRAS 703
_MSL_OS_DIRECT_MALLOC 704
_MSL_POSIX 705
_MSL_USE_NEW_FILE_APIS 704
_MSL_USE_OLD_FILE_APIS 705
_open 143
_open_osfhandle 111
_outp 49
_outpd 49
_putenv 111, 488
_RunInit 66
_searchenv 112
_setmode 184
_SetupArgs 66
_splitpath 113
_strcmpi 114
_strdate 115
_strupd 116
_strcmp 116
_stricoll 117
_strlwr 118
_strncmp 119
_strncoll 120
_strnicmp 120

```
_strnicoll 121
_strnset 122
_strev 123
_strset 123
_strspnp 124
_strupr 125
_textattr 50
_textbackground 51
_textcolor 52
_ultow 127
_wcreate 140
_wcsdup 128
_wcsicmp 129
_wcsicoll 129
_wcslwr 130
_wcsncoll 131
_wcsnicmp 132
_wcsnicoll 131
_wcsnset 133
_wcsrev 134
_wesset 134
_wcsspnp 135
_wcsupr 135
_wfopen 453
_wfreopen 453
_wherex 52
_wherey 53
_wopen 143
_wremove 454
_wrename 455
_wstrrev 136
_wtmpnam 455
_wtoi 136

acosh 243
acos 206
alloca 195
alloca 39
alloca.h 39–40
AltiVec 367, 380, 403, 421
    longjmp 280
    setjmp 281
AltiVec Extensions
    fprintf 367
    printf 403
    scanf 380, 421
ANSI C 29
Argc 63
Argv 64
asctime 550
asctime_r 552
asin 206
asinf 207
asinh 244
asinl 207
assert 41
assert.h 41–42
atan 208
atan2 209
atan2f 210
atanl 210
atanf 208
atanh 245
atanl 209
atexit 462
atof 465
atoi 466
atoi 467
atol 467
atol 467, 468
atoll 468
```

Numerics

32718
Head B
 heapmin 107

A

abort 459
abs 461
access 576
acos 205
acosf 206

B

bsearch 469
btowc 628

C

calloc 473
cbrt 246

ccommand 56
ceil 211
ceilf 212
ceill 212
cerr 293
CHAR_BIT 189
CHAR_MAX 189
CHAR_MIN 189
chdir 577
chmod 308
chsize 102
cin 293
clearerr 343
clock 552
clock_t 548
close 579
closedir 88
clrscr 57, 698
Command-line Arguments 55
console.h 55–61
copysign 247
cos 213
cosf 214
cosh 214
coshf 215
coshl 215
cosl 214
cout 293
creat 140
creat 140
crtl.h 63–66
ctime 555
ctime_r 556
ctype.h 67–80
cuserid 583
Customizing SIOUX 295
Customizing WinSIOUX 697

DBL_MAX 166
DBL_MAX_10_EXP 166
DBL_MAX_EXP 166
DBL_MIN 166
DBL_MIN_10_EXP 166
DBL_MIN_EXP 165
difftime 556
direct.h 81–83
dirent.h 85–88
div 476
div_t 91
div_t structure 476
dup 584
dup2 585

E

environ 64
Environment 151
EOF 341
erf 248
erfc 249
errno 95
errno.h 95–98
excevp 587
execl 586
execle 586
execlp 586
execv 586
execve 587
exit 477
exp 215
exp2 250
expf 217
expl 217
expm1 251
extras.h 99–137

D

Data Types
floating point environment 149
Date and time 548
DBL_DIG 165
DBL_EPSILON 166
DBL_MANT_DIG 165

F_DUPFD 141
fabs 217
fabsf 218
fabsl 218
fclose 346
fcntl 141
fcntl 141

fcntl.h 139–146
fdim 252
fdopen 348
feclearexcept 152
fegetenv 160
fegetround 158
feholdexcept 161
fenv.h 149–163
FENV_ACC 151
fenv_t 149
feof 349
feraiseexcept 154
ferror 352
fesetenv 162
fesetexceptflag 155
fesetround 159
fetestexcept 156
feupdateenv 163
fexcept_t 149
fflush 354
fgetc 356
fgetpos 358
fgets 360
fgetwc 628
fgetws 629
FILE 340
File Mode Macros
 Non Windows 307
 Windows 308
File Modes 307
filelength 102
fileno 103
float.h 165
Floating Point Classification Macros 201
Floating Point Math Facilities 205
Floating point mathematics 199
Floating-Point Exception Flags 150
floor 218
floorf 219
floorl 220
FLT_DIG 165
FLT_EPSILON 166
FLT_MANT_DIG 165
FLT_MAX 166
FLT_MAX_10_EXP 166
FLT_MAX_EXP 166
FLT_MIN 166
FLT_MIN_10_EXP 166
FLT_MIN_EXP 165
FLT_RADIX 165
FLT_ROUNDS 165
fma 253
fmax 254
fmin 255
fmod 220
fmodf 221
fmodl 221
fopen 361
fpclassify 202, 204
fprintf 365
fputc 371
fputs 373
fputwc 630
fputws 630
fread 375
free 479
freopen 378
frexp 222
frexpf 223
frexpl 223
fscanf 379
fseek 387
fsetpos 390
FSp_fopen 167
FSp_fopen.h 167–169
FSRef_fopen 168
fstat 309
ftell 391
ftime 572
fwide 392
fwprintf 631
fwrite 394
fwscanf 632

G

gamma 256
gcvt 105, 510
getc 395
getch 44, 58
getchar 397
getche 45

getcwd 588	Introduction 27–32
getegid 590	inttypes.h 171–180
getenv 480	io.h 181–185
geteuid 590	isalnum 68
getgid 590	isalpha 71
GetHandle 106	isatty 591
getlogin 589	isblank 72
getpgrp 590	iscntrl 72
getpid 590	isdigit 73
getpid 590	isfinite 203
getppid 590	isgraph 74
gets 398	isgreater 223
getuid 590	isgreaterless 224
getwc 633	isless 224
getwchar 634	islesseq 225
gmtime 558	islower 74
gmtime_r 559	isnan 203
	iso646.h 187
H	isprint 75
HUGE_VAL 243	ispunct 76
HUGE_VAL 199	isspace 76
hypot 257	isunordered 225
	isupper 77
I	iswalnum 684
ilogb 258	iswalph 685
ilogbf 258	iswblank 685
ilogbl 258	iswcntrl 686
imaxabs 174	iswdigit 686
imaxdiv 175	iswgraph 687
imaxdiv_t 172	iswlower 688
inp 46	iswprint 688
inp _d 47	iswpunct 689
Input Control String 380	iswspace 689
Input Conversion Specifiers 380	iswupper 690
inp _w 47	iswdxigit 691
InstallConsole 58	isxdigit 78
INT_MAX 190	itoa 107, 510
INT_MIN 190	itow 108, 510
INT16_C 335	
INT32_C 335	J
INT64_C 335	jmp_buf 279
INT8_C 335	
Integral limits 189	K
INTMAX_C 335	kbhit 48, 59
Intrinsic functions 32	

L

labs 481
LC_ALL 193
LC_COLLATE 193
LC_CTYPE 193
LC_MONETARY 193
LC_NUMERIC 193
LC_TIME 193
lconv structure 192
LDBL_DIG 165
LDBL_EPSILON 166
LDBL_MANT_DIG 165
LDBL_MAX 166
LDBL_MAX_10_EXP 166
LDBL_MAX_EXP 166
LDBL_MIN 166
LDBL_MIN_10_EXP 166
LDBL_MIN_EXP 165
ldexp 226
ldexpf 227
ldexpl 227
ldiv 482
ldiv_t 92
ldiv_t structure 482
lgamma 259
limits.h 189
llabs 483
lldiv 483
lldiv_t 92
LLONG_MAX 190
LLONG_MIN 190
Locale specification 191
locale.h 191–193
localeconv 192
localtime 560
localtime_r 561
log 228
log10 230
log10f 230
log10l 231
log1p 260
log2 261
logb 262
logf 229
logl 229

LONG_MAX 190
LONG_MIN 190
longjmp 280
lseek 594
itoa 109, 510

M

Mac OS X
 Extras Library 31
Macros
 floating point environment 150
makepath 110, 510
malloc.h 195–196
Marco Piovanelli 292
math.h 197–272
MB_LEN_MAX 190
mblen 485
mbsinit 637
mbstate_t 627
mbstowcs 486
mbtowc 487
memchr 512
memcmp 515
memcpy 516
memmove 517
memset 518
mkdir 311
mktime 561
modf 231, 263
modff 232
modfl 232
MSL Extras Library 30
 Mac OS X 31
MSL Flags 701–706
Multithreading 33–35

N

NaN 199, 200
nan 264
nearbyint 264
nextafter 265
NULL 325

O

offsetof 325

open 143
open 143
opendir 85
outp 49
outpd 49
Output Control String 365
Output Conversion Specifiers 365
outpw 50
rename 416
rewind 417
rewinddir 88
rint 268
rinttol 269
rmdir 596
round 270
Rounding Directions 150
roundtol 271

P

path2fss 301
 perror 400
POSIX
 naming conventions 30
pow 233
powf 234
powl 234
printf 402
process.h 275–278
ptrdiff_t 326
putc 410
putchar 412
putenv 111
puts 413
putwc 638
putwchar 639

Q

qsort 488
Quiet 200

R

raise 290
rand 489
RAND_MAX 489
rand_r 491
read 595
ReadCharsFromConsole 59
readdir 86
readdir_r 87
realloc 491
remainder 266
remove 414
RemoveConsole 60
remquo 267

S

scalb 271
scanf 420
Scanset 383
SCHAR_MAX 189
SCHAR_MIN 189
SEEK_CUR 387
SEEK_END 387
SEEK_SET 387
setbuf 425
setjmp 281
setjmp.h 279–281
setlocale 193
setvbuf 428
SHRT_MAX 189
SHRT_MIN 189
SIG_DFL 287
SIG_ERR 287
SIG_IGN 287
SIGABRT 286, 460
SIGBREAK 286
SIGFPE 286
SIGILL 286
SIGINT 286
signal 287
Signal handling 285
signal.h 285–290
Signaling 200
SIGSEGV 286
SIGTERM 286
sin 235
sinf 236
sinh 237
sinhf 238
sinhl 238

sinl 236
SIOUX 31, 292
SIOUX.h 291–304, ??–304
SIOUXHandleOneEvent 302
SIOUXSettings structure 295, 697
SIOUXSetTitle 303
size_t 326
sleep 598
snprintf 430
spawn 600
spawnl 600
spawnle 600
spawnlp 600
spawnlpe 601
spawnv 600
spawnve 600
spawnvp 601
spawnvpe 601
splitpath 113, 510
sprintf 431
sqrt 238
sqrtf 239
sqrtl 240
srand 492
sscanf 432
Standard definitions 325
Standard input/output 339
stat 312
Stat Structure
 Macintosh 306
stat.h 305–315
stdarg.h 317–321
stdbool.h 323
stddef.h 325–327
stderr 340
stdin 293, 340
stdint.h 329–335
stdio.h 337–451
stdlib.h 457–510
stdout 293, 340
strcasemp 114
strcat 518
strchr 519
strcmp 520
strcmpl 114
strcoll 193, 522
strcpy 524
strcspn 525
strdate 115
strdup 116, 542
Stream Orientation 342, 625
Streams 339
strerror 527
strerror_r 528
strftime 562
stricmp 116, 542
stricoll 117
string.h 511–543
strlen 528
strlwr 118, 542
strncasecmp 118
strncat 118
strncmp 531
strncpy 119
strncoll 120
strncpy 532
strnicmp 120, 542
strnicoll 121
strnset 122, 542
strupbrk 534
strrchr 535
strrev 123, 542
strset 123, 542
strspn 536
strspnp 124
strstr 537
strtoimax 176
strtok 539
strtold 500
strtoul 502
strtoumax 177
struct timeb 571
strupr 125, 542
strxfrm 540
swprintf 640
swscanf 641
system 505

T

tan 240

tanf 241
tanh 241
tanhf 242
tanhl 243
tanl 241
tell 125
tgmath.h 545
time 568
time.h 547–570
time_t 548
timeb 571
timeb.h 571–573
timeval structure 615
Tm Structure Members. 549
tmpfile 434
tmpnam 435
tolower 78
toupper 79
towlower 692
towupper 692
trunc 272
ttyname 601
tzname 550
tzset 569

U

UCHAR_MAX 189
 UINT16_C 335
 UINT32_C 335
 UINT64_C 335
 UINT8_C 335
 UINTMAX_C 335
ULLONG_MAX 190
 ULONG_MAX 190
ultoa 510
uname 620
ungetc 437
Unicode 342, 626
unistd.h 575–604
unix.h 125–126, 607–609
unlink 602
USHRT_MAX 190
Using SIOUX 291
Using WinSIOUX 695
utime 612

utime.h 611–616
utimes 615
utsname structure 620
utsname.h 619–620

V

va_arg 318
va_copy 318
va_end 319
va_list 317
va_start 320
Variable arguments 317
vec_calloc 505
vec_free 506
vec_malloc 507
vec_realloc 508
vfprintf 440
vfscanf 442
vfwprintf 644
vfwscanf 641
vprintf 445
vsnprintf 447
vsprintf 449
vsprintf 449
vsscanf 451
vswprintf 646
vswscanf 642
vwprintf 647
vwscanf 643

W

WASTE 292
watof 648
wchar.h 623–682
WCHAR_MAX 627
WCHAR_MIN 627
wchar_t 326
wescat 649
wcschr 650
wcscmp 650
wcscoll 651
wcscopy 652
wcscspn 652
wcscdup 128, 542, 682
wcsftime 653

wcsicmp 129, 542, 682
wcsicoll 129
wcsncmp 542, 682
wcslen 654
wcslwr 130, 542, 682
wesncat 654
wcsncmp 655
wesncoll 131
wcsncpy 656
wcsnet 133
wcsnicmp 132
wesnicoll 131
wcsnset 543, 682
wcspbrk 657
wcsppnp 543, 682
wcsrchr 657
wcsrev 134, 543, 682
wcsset 134, 543, 682
wcsspn 659
wcspnpp 135
wesstr 660
wcstod 660
wcstoimax 178
wcstok 663
wcstombs 508
wcstoumax 179
wcsupr 135, 543, 682
wcsxfrm 670
wctime 671
wctob 671
wctomb 509
wctrans 693
wctrans_t 684
wctype.h 683–693
wctype_t 684
WEOF 627
win_t 627
WinSIOUX 696
WinSIOUX.h 695–698
wmemchr 672
wmemcmp 673
wmemcpy 674
wmemmove 674
wmemset 675
wprintf 676
write 604
WriteCharsToConsole 61
wscanf 679
wstrrev 136
wtoi 510, 543, 682