

CodeWarrior™

Development Tools

MSL C Reference

Revised: <020325>

Metrowerks, the Metrowerks insignia, and CodeWarrior are registered trademarks of Metrowerks Corp. in the US and/or other countries. All other trade names, trademarks and registered trademarks are the property of their respective owners.

© Copyright. 2002. Metrowerks Corp. ALL RIGHTS RESERVED.

Metrowerks reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Metrowerks does not assume any liability arising out of the application or use of any product described herein. 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 or automobile navigation or communication, emergency systems, or other applications with a similar degree of potential hazard.

Documentation stored on electronic media may be printed for personal use only. Except for the forgoing, no portion of this documentation may be reproduced or transmitted in any form or by any means, electronic or mechanical, without prior written permission from Metrowerks.

ALL SOFTWARE, DOCUMENTATION AND RELATED MATERIALS ARE SUBJECT TO THE METROWERKS END USER LICENSE AGREEMENT FOR SUCH PRODUCT.

How to Contact Metrowerks:

Corporate Headquarters	Metrowerks Corporation 9801 Metric Blvd. Austin, TX 78758 U.S.A.
World Wide Web	http://www.metrowerks.com
Ordering & Technical Support	Voice: (800) 377-5416 Fax: (512) 997-4901

Table of Contents

1 Introduction	23
Organization of Files	23
ANSI C Standard	26
The ANSI C Library and Apple Macintosh	26
MSL Extras Library	27
POSIX Functionality	27
Console I/O and the Macintosh	27
Using Mac OS X and the Extras Library	28
Compatibility	28
Intrinsic Functions	28
2 MSL C and Multithreading	31
Introduction to Multithreading	31
Definitions	32
3 alloca.h	35
Overview of alloca.h	35
alloca	35
4 assert.h	37
Overview of assert.h	37
assert	37
5 conio.h	39
Overview of conio.h	39
clrscr	40
getch	40
getche	40
gotoxy	41
_initscr	41
inp	42
inpd	42
inpw	42
kbhit	43
outp	43
outpd	44

Table of Contents

outpw	44
_textattr	45
_textbackground	45
_textcolor	46
_wherex	46
_wherey	47
6 console.h	49
Overview of console.h	49
ccommand	50
clrscr	51
getch	52
InstallConsole	52
kbhit	52
ReadCharsFromConsole	53
RemoveConsole	53
__ttynname	54
WriteCharsToConsole	54
7 crt.h	57
Overview of crt.h	57
Argc	57
Argv	58
_DllTerminate	58
environ	58
_HandleTable	58
_CRTStartup	59
_RunInit	59
_SetupArgs	60
8 ctype.h	61
Overview of ctype.h	61
Character testing and case conversion	61
Character Sets Supported	62
isalnum	63
isalpha	65
isblank	65
iscntrl	66

isdigit	66
isgraph	67
islower	67
isprint	68
ispunct	69
isspace	69
isupper	70
isxdigit	71
tolower	71
toupper	72
9 direct.h	75
Overview of direct.h	75
_getdcwd	75
_getdiskfree	76
_getdrives	76
10 dirent.h	77
Overview of dirent.h	77
opendir	77
readdir	78
rewinddir	78
closedir	79
11 div_t.h	81
Overview of div_t.h	81
div_t	81
ldiv_t	81
lldiv_t	82
12 errno.h	83
Overview of errno.h	83
errno	83
13 extras.h	87
Overview of extras.h	87
_chdrive	89
chsiz e	90

Table of Contents

filelength	90
fileno	91
_fullpath	92
gcvt	92
_getdrive	93
GetHandle	93
_get_osfhandle	94
heapmin	94
itoa	95
itow	95
ltoa	96
_ltow	96
makepath	97
_open_osfhandle	98
putenv	98
_searchenv	99
splitpath	99
strcasecmp	100
strcmpi	101
strdate	101
strupdup	102
strcmp	102
stricoll	103
strlwr	103
strncasecmp	104
strncmpi	105
strncoll	105
strnicmp	106
strnicoll	107
strnset	108
strrev	108
strset	109
strspnp	109
strupr	110
tell	110
ultoa	111
_ultow	112

wcsdup	113
wcsicmp	113
wcsicoll	114
wcslwr	114
wcsncoll	115
wcsnicoll	116
wcsnicmp	116
wcsnset	117
wcsrev	118
wcsset	118
wcsspnp	119
wcsupr	119
wstrrev	120
wtoi	120
14 fcntl.h	123
Overview of fcntl.h	123
fcntl.h and UNIX Compatibility	123
creat	124
fcntl	125
open	126
15 fenv.h	131
Overview of fenv.h	131
Data Types.	131
fenv_t	131
fexcept_t	131
Macros	131
Floating-Point Exception Flags	132
Rounding Directions	132
Environment	132
Pragmas.	132
FENV_ACC	133
Floating-point exceptions	133
feclearexcept	133
fegetexceptflag	134
feraiseexcept	136

Table of Contents

fesetexceptflag	136
fetestexcept	137
Rounding	138
fegetround	139
fesetround	140
Environment	140
fegetenv	141
feholdexcept	141
fesetenv	143
feupdateenv	143
16 float.h	145
Overview of float.h	145
Floating point number characteristics	145
17 FSp_fopen.h	147
Overview of FSp_fopen.h	147
FSp_fopen	147
FSRef_fopen	148
FSRefParentAndFilename_fopen	148
18 inttypes.h	151
Overview of inttypes.h	151
Greatest-Width Integer Types	152
imaxdiv_t	152
Greatest-Width Format Specifier Macros	152
Greatest-Width Integer Functions	155
imaxabs	155
imaxdiv	155
strtoimax	156
strtoumax	157
wcstoimax	158
wcstoumax	159
19 io.h	161
Overview of io.h	161
_finddata_t	161
_findclose	162

<u>findfirst</u>	162
<u>findnext</u>	163
20 iso646.h	165
Overview of iso646.h	165
21 limits.h	167
Overview of limits.h	167
Integral type limits	167
22 locale.h	169
Overview of locale.h	169
Locale specification	169
localeconv	170
setlocale	171
23 malloc.h	173
Overview of malloc.h	173
alloca	173
Non Standard <malloc.h> Functions	174
24 math.h	175
Overview of math.h.	175
Floating point mathematics	178
NaN Not a Number	178
Floating point error testing.	179
Inlined Intrinsics Option	179
Floating Point Classification Macros.	179
Enumerated Constants	179
fpclassify	180
isfinite	181
isnan	181
isnormal	181
signbit	182
Floating Point Math Facilities	182
acos	182
acosf	183
acosl	183

Table of Contents

asin	184
asinf	184
asinl	185
atan	185
atanf	185
atanl	185
atan2	186
atan2f	187
atan2l	187
ceil	187
ceilf	188
ceill	188
cos	188
cosf	189
cosl	189
cosh	189
coshf	190
coshl	190
exp	190
expf	192
expl	192
fabs	192
fabsf	193
fabsl	193
floor	193
floorf	194
floorl	194
fmod	194
fmodf	195
fmodl	195
frexp	196
frexpf	197
freexpl	197
isgreater	197
isgreaterless	197
isless	198
islessequal	198

isunordered	199
ldexp	199
ldexpf	200
ldexpl	200
log	200
logf	202
logl	202
log10	202
log10f	203
log10l	203
modf	203
modff	204
modfl	204
pow	204
powf	206
powl	206
sin	206
sinf	207
sinl	207
sinh	207
sinhf	208
sinhl	208
sqrt	208
sqrtf	209
sqrtl	209
tan	209
tanf	210
tanl	211
tanh	211
tanhf	212
tanhl	212
HUGE_VAL	212
C99 Implementations	212
acosh	212
asinh	213
atanh	214
copysign	215

Table of Contents

erf	216
erfc	216
exp2	217
expm1	218
fdim	219
fmax	219
fmin	220
gamma	221
hypot	222
lgamma	223
log1p	223
log2	224
logb	225
nan	226
nearbyint	227
nextafter	227
remainder	229
remquo	230
rint	230
rinttol	231
round	232
roundtol	232
scalb	233
trunc	233
25 Process.h	235
Overview of Process.h	235
_beginthread	235
_beginthreadex	236
_endthread	237
_endthreadex	237
26 setjmp.h	239
Overview of setjmp.h	239
Non-local jumps and exception handling	239
longjmp	240
setjmp	241

27 signal.h	245
Overview of signal.h	245
Signal handling	245
signal	247
raise	249
28 SIOUX.h	251
Overview of SIOUX.	251
Using SIOUX	251
SIOUX for Macintosh	252
Creating a Project with SIOUX	253
Customizing SIOUX	255
Changing the size and location	259
Using SIOUX windows in your own application	261
path2fss	262
SIOUXHandleOneEvent	263
SIOUXSetTitle	264
29 stat.h	267
Overview of stat.h	267
stat.h and UNIX Compatibility	267
Stat Structure and Definitions	267
chmod	270
fstat	271
mkdir	272
stat	273
umask	275
30 stdarg.h	277
Overview of stdarg.h	277
Variable arguments for functions	277
va_arg	278
va_copy	278
va_end	279
va_start	279
31stdbool.h	283
Overview ofstdbool.h.	283

Table of Contents

32 stddef.h	285
Overview of stddef.h	285
Commonly used definitions	285
NULL	285
offsetof	285
ptrdiff_t	286
size_t	286
wchar_t	286
33 stdint.h	287
Overview of stdint.h	287
Integer types	287
Limits of specified-width integer types	289
Macros for integer constants	293
Macros for greatest-width integer constants	293
34 stdio.h	295
Overview of stdio.h.	295
Standard input/output	297
Streams	297
File position indicator	299
End-of-file and errors	300
Wide Character and Byte Character Stream Orientation	300
Stream Orientation and Standard Input/Output	301
clearerr	301
fclose	303
fdopen	305
feof	306
ferror	308
fflush	309
fgetc	311
fgetpos	313
fgets	315
_fileno	316
fopen	316
fprintf	320
fputc	327

Table of Contents

fputs	328
fread	330
freopen	332
fscanf	333
fseek	340
fsetpos	343
ftell	344
fwide	345
fwrite	346
getc	348
getchar	349
gets	351
perror	352
printf	353
putc	362
putchar	363
puts	364
remove	365
rename	366
rewind	368
scanf	369
setbuf	375
setvbuf	376
snprintf	378
sprintf	379
sscanf	381
tmpfile	382
tmpnam	383
ungetc	385
vfprintf	386
vfscanf	389
vprintf	391
vsnprintf	392
vsprintf	395
vsscanf	396

Table of Contents

35 stdlib.h	399
Overview of stdlib.h	399
abort	401
abs	402
atexit	403
atof	405
atoi	406
atol	407
bsearch	408
calloc	412
div	414
exit	415
free	417
getenv	417
labs	418
ldiv	419
llabs	420
lldiv	420
malloc	421
mblen	422
mbstowcs	422
mbtowc	423
_putenv	424
qsort	425
rand	426
realloc	427
srand	428
strtod	429
strtof	431
strtol	431
strtold	434
strtoll	435
strtoul	436
strtoull	438
system	439
vec_calloc	439
vec_free	440

vec_malloc	441
vec_realloc	441
wcstombs	442
wctomb	443
Non Standard <stdlib.h> Functions	444
36 string.h	445
Overview of string.h	445
memchr	446
memcmp	448
memcpy	449
memmove	450
memset	451
strcat	451
strchr	452
strcmp	453
strcoll	454
strcpy	456
strcspn	457
strerror	458
strlen	459
strncat	460
strncmp	461
strncpy	462
strpbrk	464
strrchr	465
strspn	465
strstr	467
strtok	468
strxfrm	470
Non Standard <string.h> Functions	471
37 tgmath.h	473
Overview of tgmath.h	473
38 time.h	475
Overview of time.h	475
Date and time	476

Table of Contents

Type clock_t	476
Type time_t	476
struct tm	476
tzname	478
asctime	478
clock	479
ctime	481
difftime	481
gmtime	483
localtime	484
mktime	484
strftime	485
time	491
tzset	492
Non Standard <time.h> Functions	493
 39 unistd.h	 495
Overview of unistd.h	495
unistd.h and UNIX compatibility	496
access	496
chdir	497
close	499
cuserid	502
cwait	503
dup	503
dup2	504
exec functions	505
getcwd	507
getlogin	508
getpid	508
isatty	510
lseek	511
read	512
rmdir	513
sleep	515
spawn functions	517
tbyname	518

unlink	519
write	520
40 unix.h	523
Overview of unix.h	523
UNIX Compatibility	523
_fcreator	523
_ftype	524
41 utime.h	527
Overview of utime.h	527
utime.h and UNIX Compatibility	527
utime	527
utimes	530
42 utsname.h	533
Overview of utsname.h	533
utsname.h and UNIX Compatibility	533
uname	533
43 wchar.h	537
Overview of wchar.h	537
Multibyte character functions	540
Wide Character and Byte Character Stream Orientation	540
Stream Orientation and Standard Input/Output	542
Definitions	542
btowc	543
fgetwc	543
fgetws	544
fputwc	545
fputws	545
fwprintf	546
fwscanf	547
getwc	548
getwchar	548
mbrlen	549
mbrtowc	550
mbsrtowcs	551

Table of Contents

putwc	552
putwchar	553
swprintf	553
swscanf	554
vfwscanf	555
vswscanf	556
vwscanf	557
vfwprintf	558
vswprintf	559
vwprintf	560
watof	561
wcrtomb	561
wcsncat	562
wcschr	563
wcscmp	563
wcscoll	564
wcscspn	564
wcscopy	565
wcsftime	566
wcslen	566
wcsncat	567
wcsncmp	568
wcsncpy	568
wcspbrk	569
wcsrchr	569
wcsrtombs	570
wcsspn	571
wcsstr	572
wcstod	572
wcstof	573
wcstok	574
wcstol	575
wcstold	577
wcstoll	578
wcstoul	579
wcstoull	580
wcsxfrm	582

wctime	582
wctob	583
wmemchr	583
wmemcmp	584
wmemcpy	585
wmemmove	585
wmemset	586
wprintf	586
wscanf	591
Non Standard <wchar.h> Functions	594
44 wctype.h	595
Overview of wctype.h	595
Types	596
iswalnum	596
iswalpha	597
iswblank	597
iswcntrl	598
iswdigit	598
iswgraph	599
iswlower	599
iswprint	600
iswpunct	600
iswspace	601
iswupper	601
iswxdigit	602
towctrans	602
towlower	603
towupper	603
wctrans	604
45 WinSIOUX.h	605
Overview of WinSIOUX	605
Using WinSIOUX	605
WinSIOUX for Windows.	606
Creating a Project with WinSIOUX	607
Customizing WinSIOUX	607

Table of Contents

clrscr	608
46 MSL Flags	611
Overview of the MSL Switches, Flags and Defines	611
__ANSI_OVERLOAD__	611
__MSL_IMP_EXP	612
__MSL_INTEGRAL_MATH	612
__MSL_MALLOC_0 RETURNS_NON_NULL	612
__MSL_NEEDS_EXTRAS	613
__MSL_OS_DIRECT_MALLOC	613
__MSL_CLASSIC_MALLOC	613
__MSL_USE_NEW_FILE_APIS	613
__MSL_USE_OLD_FILE_APIS	614
__MSL_POSIX	614
__SET_ERRNO__	614
Index	617

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 31 covers multithreading and thread-safeness of the Metrowerks Standard C Library functions.

The “[Overview of alloca.h](#)” on page 35 covers the non-ANSI `alloca()` function for dynamic allocation from the stack.

The “[Overview of assert.h](#)” on page 37 covers the ANSI C exception handling macro `assert()`.

The “[Overview of conio.h](#)” on page 39 covers the Windows console input and output routines.

The “[Overview of console.h](#)” on page 49 covers Macintosh console routines.

The “[Overview of crtl.h](#)” on page 57 covers Win32 console routines.

The “[Overview of ctype.h](#)” on page 61 covers the ANSI character facilities.

The “[Overview of direct.h](#)” on page 75, covers Win32x86 directory facilities.

The “[Overview of dirent.h” on page 77](#), covers various POSIX directory functions.

The “[Overview of div_t.h” on page 81](#), covers two arrays for math routines.

The “[Overview of errno.h” on page 83](#) covers ANSI global error variables.

The “[Overview of extras.h” on page 87](#) covers additional non standard functions included with the MSL library.

The “[Overview of fcntl.h” on page 123](#) covers non-ANSI control of files.

The “[Overview of fenv.h” on page 131](#), covers floating-point environment facilities.

The “[Overview of float.h” on page 145](#) covers ANSI floating point type limits.

The “[Overview of FSp_fopen.h” on page 147](#), contains Macintosh file opening routines.

The “[Overview of inttypes.h” on page 151](#) contains greatest-width integer routines and macros.

The “[Overview of io.h” on page 161](#), contains common Windows stream input and output routines.

The “[Overview of iso646.h” on page 165](#) contains operator symbol alternative words.

The “[Overview of limits.h” on page 167](#) covers ANSI integral type limits.

The “[Overview of locale.h” on page 169](#) covers ANSI character sets, numeric and monetary formats.

The “[Overview of malloc.h” on page 173](#), covers the alloca function for Windows.

The “[Overview of math.h” on page 175](#) covers ANSI floating point math facilities.

The “[Overview of Process.h](#)” on page 235, covers Windows thread process routines.

The “[Overview of setjmp.h](#)” on page 239 covers ANSI means used for saving and restoring a processor state.

The “[Overview of signal.h](#)” on page 245 covers ANSI software interrupt specifications.

The “[Overview of SIOUX](#)” on page 251 covers Metrowerks SIOUX console emulation for Macintosh.

The “[Overview of stat.h](#)” on page 267 covers non-ANSI file statistics and facilities.

The “[Overview of stdarg.h](#)” on page 277 covers ANSI custom variable argument facilities.

The “[Overview of stddef.h](#)” on page 285 covers the ANSI Standard Definitions.

The “[Overview of stdint.h](#)” on page 287 covers the latest integer types macros.

The “[Overview of stdio.h](#)” on page 295 covers ANSI standard input and output routines.

The “[Overview of stdlib.h](#)” on page 399 covers common ANSI library facilities.

The “[Overview of string.h](#)” on page 445 covers ANSI null terminated character array facilities.

The “[Overview of tgmath.h](#)” on page 473 lists type-generic math macros.

The “[Overview of time.h](#)” on page 475 covers ANSI clock, date and time conversion and formatting facilities.

The “[Overview of unistd.h](#)” on page 495 covers many of the common non-ANSI facilities.

The “[Overview of unix.h](#)” on page 523 covers some Metrowerks non-ANS facilities.

The “[Overview of utime.h](#)” on page 527 covers non-ANSI file access time facilities.

The “[Overview of utsname.h](#)” on page 533 covers the non-ANSI equipment naming facilities.

The “[Overview of wchar.h](#)” on page 537 covers the wide character set for single and array facilities.

The “[Overview of wctype.h](#)” on page 595 covers the wide character set type comparison facilities.

The “[Overview of WinSIOUX](#)” on page 605 covers Metrowerks SIOUX console emulation for Windows.

The “[Overview of the MSL Switches, Flags and Defines](#)” on page 611 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.

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 613](#), 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 27](#) are included in the MSL Extras Library.

Windows Only These extended functions are enabled as the same name with a leading underscore.

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			
Windows Only	conio.h	direct.h	io.h

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.

Introduction

Intrinsic Functions

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

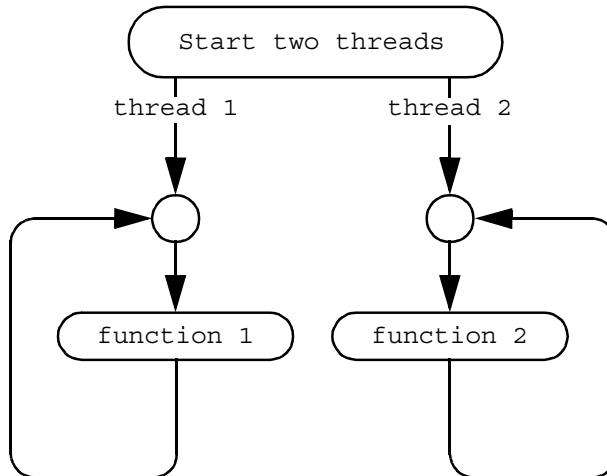
Essentially, there are no standards for implementing multithreading. In particular, neither the C99 Standard nor the POSIX Standard makes reference to threads. For the MSL C library, we define thread-safety as:

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 thread-safe:

Table 2.1 Thread-safe MLS C Functions

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
realloc	scanf	setbuf	setlocale	setvbuf	signal
strrand	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.

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 35](#) that allocates memory from the stack

alloca

Allocates memory quickly on the stack.

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

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

Parameters Parameters for this function are:

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.

alloca.h

Overview of alloca.h

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

See Also [“calloc” on page 412](#)
[“free” on page 416](#)
[“malloc” on page 421](#)
[“realloc” on page 427\)](#)

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 37](#), that outputs a diagnostic message and stops the program if a test fails.

assert

Abort a program if a test is false.

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

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

Parameters Parameters for this function are:

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.

See Also [“abort” on page 401.](#)

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 40](#) clears the console window
- [“`getch`” on page 40](#) reads a char from the input screen
- [“`getche`” on page 40](#) reads a char and echoes to the screen
- [“`gotoxy`” on page 41](#) places the cursor on a console window
- [“`initscr`” on page 41](#) sets up a console in a GUI application
- [“`inp`” on page 41](#) reads a byte from an input port
- [“`inpд`” on page 42](#) reads a double word from an input port
- [“`inpw`” on page 42](#) reads a word from an input port
- [“`kbhit`” on page 43](#) returns true if a key is pressed
- [“`outp`” on page 43](#) outputs a byte to a port
- [“`outpd`” on page 44](#) outputs a double word to a port
- [“`outpw`” on page 44](#) outputs a word to a port
- [“`textattr`” on page 45](#) sets the text attributes
- [“`textbackground`” on page 45](#) sets the text background color
- [“`textcolor`” on page 46](#) sets the text color
- [“`wherex`” on page 46](#) returns the horizontal coordinate
- [“`wherey`” on page 46](#) returns the vertical coordinate

_clrscr

Clears the standard output screen.

Compatibility This function is Windows only when declared from this header.

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

Parameters This facility has no parameters.

Return No value is returned in this implementation.

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.

Compatibility This function is Windows only when declared from this header.

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

Parameters This facility has no parameters.

The function `getch` returns the char read.

See Also [“getc” on page 347](#)

[“getchar” on page 349](#)

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.

Compatibility This function is Windows only when declared from this header.

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

Parameters This facility has no parameters.

Return The function `getche` returns the char read.

See Also [“getc” on page 347](#)

[“getchar” on page 349](#)

_gotoxy

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

Compatibility This function is Windows only when declared from this header.

Prototype

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

Parameters Parameters for this facility are:

x	int	vertical screen coordinate
y	int	horizontal screen coordinate

Return No value is returned in this implementation.

See Also [“_wherex” on page 46](#)
[“_wherey” on page 46](#)

_initscr

Sets up standard 80x25 console with no scrolling region.

Compatibility This function is Windows only when declared from this header.

Prototype

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

Parameters This facility has no parameters.

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

Return No value is returned in this implementation.

inp

Reads a byte from the specified port

Compatibility This function is Windows only when declared from this header.

Prototype

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

Parameters Parameters for this facility are:

port unsigned short a port specified by number

Return The value as a byte read is returned.

See Also [“inp” on page 41](#)

[“inpw” on page 42](#)

inp

Reads a double word from the specified port

Compatibility This function is Windows only when declared from this header.

Prototype

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

Parameters Parameters for this facility are:

port unsigned short a port specified by number

Return The value as a long read is returned.

See Also [“inp” on page 41](#)

[“inpw” on page 42](#)

inpw

Reads a word from the specified port

Compatibility This function is Windows only when declared from this header.

Prototype

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

Parameters Parameters for this facility are:

port unsigned short a port specified by number

Return The value as a word read is returned.

See Also [“inp” on page 41](#)

[“inp” on page 42](#)

kbhit

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

Compatibility This function is Windows only when declared from this header.

Prototype

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

Parameters 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

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

See Also [“getch” on page 40](#)

[“getche” on page 40](#)

outp

Outputs a byte to a specified port.

Compatibility This function is Windows only when declared from this header.

Prototype

```
#include <conio.h>
void outp (unsigned short pt, unsigned char out);
void _outp (unsigned short pt, unsigned char )out;
```

Parameters Parameters for this facility are:

port unsigned short a port specified by number

out unsigned char a byte value sent to the output

Return No value is returned in this implementation.

See Also [“outpd” on page 44](#)

[“outpw” on page 44](#)

outpd

Description

Compatibility This function is Windows only when declared from this header.

Prototype

```
#include <conio.h>
void outpd (unsigned short pt, unsigned long out);
void _outpd (unsigned short pt, unsigned long out);
```

Parameters Parameters for this facility are:

pt unsigned short a port specified by number

out unsigned long a double word value sent to the output

Return No value is returned in this implementation.

See Also [“outp” on page 43](#)

[“outpw” on page 44](#)

outpw

Description

Compatibility This function is Windows only when declared from this header.

Prototype

```
#include <conio.h>
void outpw (unsigned short pt, unsigned short out);
void _outpw (unsigned short pt, unsigned short
out);
```

Parameters Parameters for this facility are:

pt unsigned short a port specified by number

out unsigned long a word value sent to the output

Return No value is returned in this implementation.

See Also [“outp” on page 43](#)

[“outpd” on page 44](#)

_textattr

This function sets the attributes for the console text.

Compatibility This function is Windows only when declared from this header.

Prototype `#include <conio.h>`
`void _textattr(int newattr);`

Parameters Parameters for this facility are:

`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`.

Return No value is returned in this implementation.

See Also [“textbackground” on page 45](#)
[“textcolor” on page 46](#)

_textbackground

This function sets the console's background color.

Compatibility This function is Windows only when declared from this header.

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

Parameters Parameters for this facility are:

`newcolor` int the background color to be set

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

Return No value is returned in this implementation.

See Also [“textattr” on page 45](#)
[“textcolor” on page 46](#)

_textcolor

This function sets the console's text color.

Compatibility This function is Windows only when declared from this header.

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

Parameters Parameters for this facility are:

`newcolor` int the text color to be set

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

Return No value is returned in this implementation.

See Also [“textattr” on page 45](#)

[“textbackground” on page 45](#)

_wherex

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

Compatibility This function is Windows only when declared from this header.

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

Parameters This facility has no parameters.

Return Returns the horizontal coordinate.

See Also [“wherey” on page 46](#)

[“gotoxy” on page 41](#)

_wherey

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

Compatibility This function is Windows only when declared from this header.

Prototype `#include <conio.h>`

```
int _wherex(void);
```

Parameters This facility has no parameters.

Return Returns the horizontal coordinate.

See Also [“_gotoxy” on page 41](#)

[“_wherex” on page 46](#)

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 consist 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 50](#), which helps you port a program that relies on command-line arguments.
- [“clrscr” on page 51](#), clears the SIOUX window and flushes the buffer.
- [“getch” on page 52](#) returns the keyboard character pressed when an ascii key is pressed
- [“InstallConsole” on page 52](#) installs the Console package.
- [“kbhit” on page 52](#) returns true if any keyboard key is pressed without retrieving the key
- [“ReadCharsFromConsole” on page 53](#) reads from the Console into a buffer.
- [“RemoveConsole” on page 53](#) removes the console package.
- [“_ttyname” on page 54](#) Returns the name of the terminal associated with the file id. The unix.h function ttyname calls this function
- [“WriteCharsToConsole” on page 54](#) writes a stream of output to the Console window.

ccommand

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

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

Prototype

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

Parameters Parameters for this function are:

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

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

Remarks This function displays a dialog that lets you enter arguments and redirect standard input and output. Please refer to [“Overview of SIOUX” on page 251](#), for information on customizing SIOUX, or setting console options.

NOTE 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 51](#) has an example of command line usage

Return This function returns the number of arguments you entered.

See Also [“Customizing SIOUX” on page 255](#)

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;

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

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

Parameters None

Remarks This function is used to select all and clear the screen and buffer by calling `SIOUXclrscr` from `SIOUX.h` on the mac or `WinSIOUXclrscr` from `WinSIOUX.h` for Windows.

getch

Returns the keyboard character pressed when an ascii key is pressed

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

Prototype

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

Parameters None

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

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

See Also [“kbhit” on page 52](#)

InstallConsole

Installs the Console package.

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

Prototype

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

Parameters Parameters for this function are:

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.

Returns Returns any error

See Also [“Customizing SIOUX” on page 255](#)
[“RemoveConsole” on page 53](#)

kbhit

Returns true if any keyboard key is pressed.

Compatibility	Macintosh only, This function may not be implemented on all Mac OS versions.
Prototype	#include <console.h> int kbhit(void);
Parameters	None
Remarks	Returns true if any keyboard key is pressed without retrieving the key used for stopping a loop by pressing any key
Returns	Returns non zero when any keyboard key is pressed.
See Also	“getch” on page 52

ReadCharsFromConsole

Reads from the Console into a buffer.

Compatibility	Macintosh only, This function may not be implemented on all Mac OS versions.
Prototype	#include <console.h> extern long ReadCharsFromConsole (char *buffer, long n);
Parameters	Parameters for this function are: 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.
Returns	Returns any error.
See Also	“WriteCharsToConsole” on page 54

RemoveConsole

Removes the console package.

Compatibility	Macintosh only, This function may not be implemented on all Mac OS versions.
Prototype	#include <console.h> extern void RemoveConsole(void);

Parameters	None
Remarks	Removes the console package. It is called after all other streams are closed and exit functions (installed by either <code>atexit</code> or <code>_atexit</code>) have been called.
Returns	Since there is no way to recover from an error, this function doesn't need to return any.
See Also	“Customizing SIOUX” on page 255 “InstallConsole” on page 52

__ttyname

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

Compatibility	Macintosh only, This function may not be implemented on all Mac OS versions.
Prototype	#include <console.h> extern char * __ttyname(long fildes);
Parameters	Parameters for this function are: fildes long The file descriptor
Remarks	Returns the name of the terminal associated with the file id. The unix.h function ttyname calls this function (we need to map the int to a long for size of int variance).
Returns	Returns the name of the terminal associated with the file id.
See Also	“ttyname” on page 518

WriteCharsToConsole

Writes a stream of output to the Console window.

Compatibility	Macintosh only, This function may not be implemented on all Mac OS versions.
Prototype	#include <console.h> a. extern long WriteCharsToConsole (char *buffer, long n);

Parameters	Parameters for this function are:		
	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.		
Returns	Returns any error		
See Also	“ReadCharsFromConsole” on page 53		

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 57](#), is the argument list count
- [“Argv” on page 58](#), the argument list variables
- [“_DllTerminate” on page 58](#), shows when a DLL is running terminate code.
- [“environ” on page 58](#), is the environment pointers
- [“_HandleTable” on page 58](#), is a structure allocated for each ed file handle
- [“_CRTStartup” on page 59](#), initializes the C Runtime start-up routines.
- [“_RunInit” on page 59](#), initializes the runtime, static classes and variables.
- [“_SetupArgs” on page 60](#), sets up the command line arguments.

Argc

The argument count variable

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

Prototype

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

Remarks Used for command line argument count.

Argv

The argument command variables.

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

Prototype

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

Remarks The command line arguments.

_DllTerminate

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

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

Prototype

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

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

environ

The environment pointers

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

Prototype

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

Remarks This is a pointer to the environment.

_HandleTable

FileStruct is a structure allocated for each file handle

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

Prototype

```
#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.

_CRTStartup

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

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

Prototype

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

Parameters None

_RunInit

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

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

Prototype

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

Parameters None

_SetupArgs

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

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

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

Parameters None

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 63](#) tests for alphabetical and numerical characters
- [“`isalpha`” on page 65](#) tests for alphabetical characters
- [“`isblank`” on page 65](#) tests for a space between words and is dependent upon the locale usage
- [“`iscntrl`” on page 66](#) tests for control characters
- [“`isdigit`” on page 66](#) tests for digit characters
- [“`isgraph`” on page 67](#) tests for graphical characters
- [“`islower`” on page 67](#) tests for lower case characters
- [“`isprint`” on page 68](#) tests for printable characters
- [“`ispunct`” on page 69](#) tests for punctuation characters
- [“`isspace`” on page 69](#) tests for white space characters
- [“`isupper`” on page 70](#) test for upper case characters

- [“`isxdigit`” on page 71](#) texts for hexadecimal characters
- [“`tolower`” on page 71](#) changes from uppercase to lowercase
- [“`toupper`” on page 72](#) 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 62](#) for return values.

Table 8.1 Character testing functions

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

isalnum

Determine character type.

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

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

Parameters Parameters for this facility are:

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 63](#).

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

Return [Table 8.1](#) describes what the character testing functions return.

See Also [“tolower” on page 71](#)
[“toupper” on page 72](#)

Listing 8.1 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]) ?
```

ctype.h

Overview of ctype.h

```
    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.

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

Prototype

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

Parameters Parameters for this facility are:

c int character being evaluated

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

Return [“Character testing functions” on page 62](#) describes what the character testing functions return.

Listing 8.2 For example usage

For example usage see [“Character testing functions example” on page 63](#)

isblank

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

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

Prototype

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

Parameters Parameters for this facility are:

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.

Return [Table 8.1](#) describes what the character testing functions return.

See Also [“`isspace`” on page 69](#)

iscntrl

Determine character type.

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

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

Parameters Parameters for this facility are:

c int character being evaluated

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

Return [“Character testing functions” on page 62](#) describes what the character testing functions return.

Listing 8.3 For example usage

For example usage see [“Character testing functions example” on page 63](#)

isdigit

Determine character type.

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

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

Parameters Parameters for this facility are:

	c	int	character being evaluated
Remarks	This macro returns nonzero for true, zero for false, depending on the integer value of c.		
Return	“Character testing functions” on page 62 describes what the character testing functions return.		

Listing 8.4 For example usage

For example usage see [“Character testing functions example” on page 63](#)

isgraph

Determine character type.

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

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

Parameters
Parameters for this facility are:

c int character being evaluated

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

Return
[“Character testing functions” on page 62](#) describes what the character testing functions return.

Listing 8.5 For example usage

For example usage see [“Character testing functions example” on page 63](#)

islower

Determine character type.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <ctype.h> int islower(int c);
Parameters	Parameters for this facility are: 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, <code>islower</code> returns true only for the lowercase characters.
Return	"Character testing functions" on page 62 describes what the character testing functions return.

Listing 8.6 For example usage

For example usage see ["Character testing functions example" on page 63](#)

isprint

Determine character type.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <ctype.h> int isprint(int c);
Parameters	Parameters for this facility are: c int character being evaluated
Remarks	This macro returns nonzero for true, zero for false, depending on the integer value of c.
Return	"Character testing functions" on page 62 describes what the character testing functions return.

Listing 8.7 For example usage

For example usage see "[Character testing functions example](#)" on [page 63](#)

ispunct

Determine character type.

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

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

Parameters Parameters for this facility are:

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.

Return ["Character testing functions" on page 62](#) describes what the character testing functions return.

Listing 8.8 For example usage

For example usage see "[Character testing functions example](#)" on [page 63](#)

isspace

Determine character type.

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

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

Parameters Parameters for this facility are:

	c	int	character being evaluated
Remarks	This macro returns nonzero for true, zero for false, depending on the integer value of <i>c</i> .		
	In the "C" locale, <code>isspace</code> returns <code>true</code> only for the standard white-space characters.		
Return	"Character testing functions" on page 62 describes what the character testing functions return.		

Listing 8.9 For example usage

For example usage see ["Character testing functions example" on page 63](#)

isupper

Determine character type.

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

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

Parameters Parameters for this facility are:

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.

Return ["Character testing functions" on page 62](#) describes what the character testing functions return.

Listing 8.10 For example usage

For example usage see ["Character testing functions example" on page 63](#)

isxdigit

Determine hexadecimal type.

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

Prototype

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

Parameters Parameters for this facility are:

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 63

Return “[Character testing functions](#)” on page 62 describes what the character testing functions return.

Listing 8.11 For example usage

For example usage see “[Character testing functions example](#)” on page 63

tolower

Character conversion macro. For example usage see “[Example of tolower\(\), toupper\(\) usage.](#)” on page 72.

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

Prototype

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

Parameters Parameters for this facility are:

c int character being evaluated

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

See Also [“isalpha” on page 65](#)

[“toupper” on page 72](#).

Listing 8.12 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.

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

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

Parameters Parameters for this facility are:

`c` `int` character being evaluated

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

See Also ["isalpha" on page 65](#)
 ["tolower" on page 71](#)

Listing 8.13 For example usage

see ["Example of tolower\(\), toupper\(\) usage." on page 72](#)

ctype.h

Overview of ctype.h

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 75](#) gets the current working drive
- [“`_getdiskfree`” on page 76](#) gets the amount of free disk space
- [“`_getdrives`” on page 76](#) gets drives available

`_getdcwd`

Determines the current working directory information.

Compatibility This function is Windows only when declared from this header.

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

Parameters Parameters for this function are:

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.

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

See Also [“`chdrive`” on page 89](#)

_getdiskfree

Determines the free disk space.

Compatibility This function is Windows only when declared from this header.

Prototype

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

Parameters Parameters for this function are:

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.

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

See Also [“_getdrive” on page 93](#)

_getdrives

Determines the logical drive information.

Compatibility This function is Windows only when declared from this header.

Prototype

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

Parameters There is no parameter for this function.

Return 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.

See Also [“_getdrive” on page 93](#)

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”](#) opens a directory stream.
- [“readdir”](#) reads a directory stream.
- [“rewinddir”](#) rewinds a directory stream.
- [“closedir”](#) 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” on page 27](#), for information on POSIX naming conventions.

opendir

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

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

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

Parameters Parameters for this function are:

	path	const char *	The path of the directory to be opened
Return	This function returns <code>NULL</code> if the directory can not be opened. If successful a directory stream pointer of type <code>DIR *</code> is returned.		
See Also	“closedir” on page 79		

readdir

This function is used read the next directory entry.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <dirent.h> struct dirent * readdir(DIR *dp);</pre>
Parameters	Parameters for this function are:
	dp DIR * The stream being read
Remarks	The data pointed to by <code>readdir()</code> may be overwritten by another call to <code>readdir()</code> .
Return	The function <code>readdir</code> returns the next directory entry from the stream <code>dp</code> as a pointer of <code>struct dirent</code> .
See Also	“rewinddir” on page 78

rewinddir

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

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <dirent.h> void rewinddir(DIR * dp);</pre>
Parameters	Parameters for this function are:

dp DIR * The stream being rewound

Return There is no return.

See Also [“readdir” on page 78](#)

closedir

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

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

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

Parameters Parameters for this function are:

dp DIR * The directory stream pointer to be closed

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

See Also [“opendir” on page 77](#)

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 81](#), stores remainder and quotient variables
- [“ldiv_t” on page 81](#), stores remainder and quotient variables
- [“lldiv_t” on page 82](#) stores remainder and quotient variables

div_t

Stores the remainder and quotient from the div function.

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

Prototype

```
#include <div_t.h>
typedef struct {
    int quot;
    int rem;
} div_t;
```

See Also [“div” on page 414](#)

ldiv_t

Stores the remainder and quotient from the ldiv function.

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

div_t.h

Overview of div_t.h

Prototype `#include <div_t.h>`
 `typedef struct {`
 `int quot;`
 `int rem;`
 } ldiv_t;

See Also [“ldiv” on page 419](#)

lldiv_t

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

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

Prototype `#include <div_t.h>`
 `typedef struct {`
 `long long quot;`
 `long long rem;`
 } lldiv_t;

See Also [“ldiv” on page 419](#)

errno.h

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

Overview of `errno.h`

There is one global declared in `errno.h`

- [“`errno`” on page 83](#)

`errno`

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

Compatibility This macro may not be implemented on all platforms.

Prototype `#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 84.

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 12.1](#).

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.

Table 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 <code>__MacOSErrNo</code>

Unassigned errno	reserved for future use
E2BIG	Argument list too long
EAGAIN	Resource temporarily unavailable
EBUSY	Device busy
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

errno.h

Overview of errno.h

ESPIPE	Illegal seek
ESRCH	No such process
EUNKNOWN	Unknown error
EXDEV	Cross-device link

Listing 12.1 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 89](#) changes the working drive
- [“chsize” on page 90](#) changes a file size
- [“filelength” on page 90](#) gets the file length
- [“fileno” on page 91](#) gets the file number
- [“ fullname” on page 92](#) gets the full pathname
- [“gcvt” on page 92](#) converts a floating point value to a string
- [“ getdrive” on page 93](#) gets the drive as a number
- [“GetHandle” on page 93](#) Windows get console handle
- [“_get_osfhandle” on page 94](#) get an operating system handle
- [“heapmin” on page 94](#) releases unused heap to the system
- [“itoa” on page 95](#) int to string
- [“itow” on page 95](#) int to wide character string
- [“ltoa” on page 96](#) long to string
- [“_ltow” on page 96](#) long to wide character string
- [“makepath” on page 97](#) creates a path
- [“open_osfhandle” on page 98](#) open an operating system handle
- [“putenv” on page 98](#) put an environment variable
- [“searchenv” on page 99](#) search the environment variable

- [“`splitpath`” on page 99](#) splits a path into components
- [“`strcasecmp`” on page 100](#) string ignore case compare
- [“`strcmpl`” on page 101](#) case insensitive string compare
- [“`strdate`” on page 101](#) stores a date in a string
- [“`strdup`” on page 102](#) duplicates a string
- [“`stricmp`” on page 102](#) case insensitive string compare
- [“`stricoll`” on page 103](#) locale collating string comparison
- [“`strlwr`” on page 103](#) string to lower case
- [“`strncasecmp`” on page 104](#) string case compare with length specified
- [“`strncmpi`” on page 105](#) case insensitive string compare
- [“`strnicmp`” on page 106](#) case insensitive string compare with length specified.
- [“`strncoll`” on page 105](#) length limited comparison of a string collated by locale setting
- [“`strnicoll`” on page 107](#) case insensitive string comparison by locale.
- [“`strnset`” on page 108](#) sets a number of characters in a string
- [“`strrev`” on page 108](#) reverses characters in a string
- [“`strset`” on page 109](#) sets characters in a set
- [“`strspnp`” on page 109](#) finds a string in another string
- [“`strupr`” on page 110](#) string to upper case
- [“`tell`” on page 110](#) gets the file indicator position
- [“`ultoa`” on page 111](#) unsigned long to string
- [“`ultow`” on page 112](#) unsigned long to lower wide character string
- [“`wcsdup`” on page 113](#) wide character string duplicate
- [“`wcsicoll`” on page 114](#) case insensitive string comparison collated by locale setting
- [“`wcsicmp`” on page 113](#) wide character case insensitive string compare
- [“`wcslwr`” on page 114](#) wide character string to lower case
- [“`wcsncoll`” on page 115](#) wide character string comparison collated by locale settings

- [“wcsnicmp” on page 116](#), wide character case insensitive string compare for a length limit
- [“wcsnicoll” on page 116](#) wide character string comparison case insensitive collated by locale
- [“wcsnset” on page 117](#) sets a number of characters in a wide character string
- [“wcsrev” on page 118](#) reverses a wide character string
- [“wcsset” on page 118](#) sets characters in a wide character string
- [“wcsspnp” on page 119](#) finds a wide character string in another wide character string
- [“wstrrev” on page 120](#) reverses wide character string
- [“wcsupr” on page 119](#) wide string to upper case
- [“wtoi” on page 120](#) 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” on page 27](#), for information on POSIX naming conventions.

_chdrive

Changes the current drive by number.

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

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

Parameters Parameters for this function are:

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.

Return Zero is returned on success and negative one on failure.

See Also [“_getdrive” on page 93](#)

chsize

This function is used to change a files size.

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

Prototype

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

Parameters Parameters for this function are:

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.

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

See Also [“GetHandle” on page 93](#)

filelength

Retrieves the file length based on a file handle.

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

Prototype

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

Parameters Parameters for this function are:

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

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

See Also [“GetHandle” on page 93](#)

fileno

Obtains the file descriptor associated with a stream.

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

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

Parameters Parameters for this facility are:

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:

Table 13.1 File Descriptors for the Standard I/O Streams

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

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

See Also [“fdopen” on page 305](#)
[“open” on page 126](#)

Figure 13.1 Example of fileno() usage.

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

int main(void)
{
```

```
printf("The handle for the standard input device is: %d",
       fileno(stdin) );

    return 0;
}
Result
The handle for the standard input device is: 0
```

_fullpath

Converts a relative path name to a full path name.

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

Prototype

```
#include <extras.h>
char *_fullpath(char * absPath,
                 const char * relPath, size_t maxLength);
```

Parameters Parameters for this function are:

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.

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

gcvt

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

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

Prototype

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

```
char *_gcvt(double value,  
           int digits, char *buffer);
```

Parameters	Parameters for this function are:		
	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.		
Return	This function returns a pointer to the <code>buffer</code> argument.		
See Also	“atof” on page 405,		

_getdrive

Finds the current drive as a number.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <extras.h> int _getdrive();
Parameters	This facility has no parameters.
Return	The current drive number is returned.
See Also	“chdrive” on page 89 “GetHandle” on page 93

GetHandle

GetHandle retrieves the current objects handle.

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

Prototype	<pre>#include <extras.h> int GetHandle();</pre>
Parameters	This facility has no parameters.
Return	The device handle is returned on success. A negative one is returned on failure.
See Also	“fileno” on page 91

_get_osfhandle

Retrieve an operating system file handle.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <extras.h> long _get_osfhandle(int filehandle);</pre>
Parameters	Parameters for this function are:
	filehandle int An operating system file handle
Return	An operating system file handle as opposed to C file handle is returned if successful otherwise sets <code>errno</code> and returns NULL.
See Also	“GetHandle” on page 93 “open_osfhandle” on page 98.

heapmin

This function releases the heap memory back to the system.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <io.h> int heapmin(void); int _heapmin(void);</pre>
Parameters	This facility has no parameters.

Return 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.

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

Prototype

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

Parameters Parameters for this function are:

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.

Return A pointer to the converted string is returned.

See Also [“atoi” on page 406](#)
[“ltoa” on page 96](#)

itow

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

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

Prototype

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

Parameters Parameters for this function are:

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.

Return A pointer to the converted wide character string is returned.

See Also [“itoa” on page 95](#)

Itoa

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

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

Prototype

```
#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.

Return A pointer to the converted wide character string is returned.

See Also [“itoa” on page 95](#)

[“atol” on page 407](#)

_ltow

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

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

Prototype

```
#include <extras.h>
wchar_t *_ltow(unsigned long val,
```

```
wchar_t *str, int radix);
```

Parameters	Parameters for this function are:	
	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.	
Return	A pointer to the converted wide character string is returned.	
See Also	“itow” on page 95, “ltoa” on page 96,	

makepath

Makepath is used to create a path.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <io.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);

Parameters	Parameters for this function are:	
	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
Return	None		
See Also	“chdrive” on page 89		

_open_osfhandle

Opens an operating system handle.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <extras.h> int _open_osfhandle(long ofshandle, int flags);
Parameters	Parameters for this function are:
	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.
Return	The file handle value is returned on success. A negative one is returned on failure.
See Also	“get_osfhandle” on page 94.

putenv

Adds a string to the environmental variable.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <extras.h> int putenv(const char * inVarName) int _putenv(const char * inVarName)

Parameters	Parameters for this function are:	
	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.	
Return	Zero is returned on success or negative one on failure.	

_searchenv

Searches the environmental path for a file.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <extras.h> void _searchenv(const char *filename, const char *varname, char *pathname);	
Parameters	Parameters for this function are:	
	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.	
Return	There is no return value.	

splitpath

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

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

Prototype	#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)		
Parameters	Parameters for this function are:		
	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		
See Also	“chdir” on page 497		

strcasecmp

Ignore case string comparison function

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <string.h> int strcasecmp (const char *str1, const char *str2);
Parameters	Parameters for this function are:

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

Remarks	The function converts both strings to lower case before comparing them.
Return	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.
See Also	"strcasecmp" on page 104 "strcmp" on page 102.

strcmpl

A case insensitive string compare.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <extras.h> int strcmpl(const char *s1, const char *s2); int _strcmpl(const char *s1, const char *s2);
Parameters	Parameters for this function are: s1 const char * The first string to compare s2 const char * The comparison string
Return	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.
See Also	"strcasecmp" on page 100 "strnicmp" on page 106

strdate

The strdate function stores a date in a buffer provided.

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

Prototype	#include <time.h> char * strdate(char *str); char * _strdate(char *str);
Parameters	Parameters for this function are: str char * A char string to store the date
Return	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.
See Also	“strftime” on page 485

strup

Creates a duplicate string in memory.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <extras.h> char * strdup(const char *str); char * _strdup(const char *str);
Parameters	Parameters for this function are: str const char * The string to be copied
Return	A pointer to the storage location or NULL if unsuccessful.
See Also	“memcpy” on page 449

stricmp

A function for string comparison ignoring case.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <extras.h> int stricmp(const char *s1,const char *s2); int _stricmp(const char *s1,const char *s2);

Parameters	Parameters for this function are:
	s1 const char * The string being compared
	s2 const char * The comparison string
Return	Stricmp 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.
See Also	"strcmp" on page 453 "strncmp" on page 461

stricoll

A case insensitive locale collating string comparison.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <extras.h> int stricoll(const char *s1, const char *s2); int _stricoll(const char *s1, const char *s2);
Parameters	Parameters for this function are:
	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 <code>LC_COLLATE</code> component of the current locale.
Return	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	"strncoll" on page 105 "wcsicoll" on page 114

strlwr

This function converts a string to lowercase.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <extras.h> char * strlwr(char *str); char * _strlwr(char *str);</pre>
Parameters	Parameters for this function are:
	str char The string being converted
Return	A pointer to the converted string.
See Also	“strupr” on page 110 “tolower” on page 71

strncasecmp

	Ignore case string comparison function with length specified.
Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <string.h> int strncasecmp(const char *s1, const char *s2, unsigned n);</pre>
Parameters	Parameters for this function are:
	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.
Return	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.
See Also	“strcasecmp” on page 100

[“strnicmp” on page 106.](#)

strncmpi

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

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

Prototype

```
#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);
```

Parameters Parameters for this function are:

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.

Return 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.

See Also

[“strcmp” on page 102](#)
[“strncasecmp” on page 104](#)
[“wcscmp” on page 113](#)

strncoll

A length limited locale collating string comparison.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.									
Prototype	<pre>#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);</pre>									
Parameters	Parameters for this function are:									
	<table><tr><td>s1</td><td>const char *</td><td>The string to compare</td></tr><tr><td>s2</td><td>const char *</td><td>A comparison string</td></tr><tr><td>sz</td><td>size_t</td><td></td></tr></table>	s1	const char *	The string to compare	s2	const char *	A comparison string	sz	size_t	
s1	const char *	The string to compare								
s2	const char *	A comparison string								
sz	size_t									
Remarks	The comparison is done according to a collating sequence specified by the <code>LC_COLLATE</code> component of the current locale.									
Return	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	“strnicoll” on page 107 “wcsncoll” on page 115									

strnicmp

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

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <extras.h> iint strnicmp(const char *s1,const char *s2,int n); nt _strnicmp(const char *s1,const char *s2,int n);</pre>
Parameters	Parameters for this function are:

s1	const char *	The string being compared
s2	const char *	The comparison string
n	int	Maximum comparison length
Return	The function strnicmp 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.	
See Also	“strcmp” on page 453 “strncmp” on page 461	

strnicoll

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

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.									
Prototype	<pre>#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);</pre>									
Parameters	Parameters for this function are:									
	<table><tr><td style="vertical-align: top; padding-right: 20px;">s1</td><td>const char *</td><td>The string to compare</td></tr><tr><td style="vertical-align: top; padding-right: 20px;">s1</td><td>const char *</td><td>A comparison string</td></tr><tr><td style="vertical-align: top; padding-right: 20px;">sz</td><td>size_t</td><td></td></tr></table>	s1	const char *	The string to compare	s1	const char *	A comparison string	sz	size_t	
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 <code>LC_COLLATE</code> component of the current locale.									
Return	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 103 “wcsicoll” on page 114									

strnset

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

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

Prototype

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

Parameters Parameters for this function are:

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.

Return A pointer to the altered string is returned.

See Also [“strset” on page 109](#)

strrev

This function reverses a string.

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

Prototype

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

Parameters Parameters for this function are:

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

Return A pointer to the reversed string.

See Also [“strcpy” on page 456](#)

strset

This function sets characters of string to a character

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

Prototype

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

Parameters Parameters for this function are:

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

Return A pointer to the altered string is returned.

See Also [“strnset” on page 108](#)

strspnp

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

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

Prototype

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

Parameters Parameters for this function are:

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.

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

See Also [“strcspn” on page 457](#)

[“strspn” on page 465](#)

strupr

The function `strupr` converts a string to uppercase.

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

Prototype

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

Parameters Parameters for this function are:

str char The string being converted

Return A pointer to the converted string.

See Also [“toupper” on page 72](#)

[“strlwr” on page 103](#)

tell

Returns the current offset for a file.

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

Prototype

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

Parameters Parameters for this facility are:

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.

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

See Also [“ftell” on page 343](#)

[“lseek” on page 511](#)

Listing 13.1 Example of read() usage.

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

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

Prototype

```
#include <extras.h>
char * ultoa(unsigned long val,
             char *str, int radix);
char * _ultoa(unsigned long val,
              char *str, int radix);
```

Parameters for this function are:

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.

Return A pointer to the converted string is returned.

See Also [“ltoa” on page 96](#)

[“itoa” on page 95](#)

_ultow

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

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

Prototype

```
#include <extras.h>
wchar_t *_ultow(unsigned long val,
                 wchar_t *str, int radix);
```

Parameters Parameters for this function are:

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.

Return A pointer to the converted wide character string is returned.

See Also	“strtoul” on page 436 “itow” on page 95 “_ltow” on page 96
----------	--

wcsdup

Creates a duplicate wide character string in memory.

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

Prototype

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

Parameters Parameters for this function are:

str const wchar_t * The string to be copied

Return A pointer to the storage location or NULL if unsuccessful.

See Also	“strup” on page 102
----------	-------------------------------------

wcsicmp

A function for wide character string comparison ignoring case.

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

Prototype

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

Parameters Parameters for this function are:

s1 const wchar_t * The string being compared

s2 const wchar_t * The comparison string

Return The function _wcsicmp 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.

See Also	“strcmp” on page 453 “strncmp” on page 461
----------	---

wcsicoll

A case insensitive locale collating wide character string comparison.

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

Prototype

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

Parameters

Parameters for this function are:

- | | | |
|----|-----------------|------------------------------------|
| 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.

Return

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	“wcsncoll” on page 115 “stricoll” on page 103
----------	--

wcslwr

This function converts a string to lowercase.

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

Prototype

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

Parameters	Parameters for this function are:	
	str	wchar_t The string being converted
Return	A pointer to the converted string.	
See Also	“strupr” on page 110 “strlwr” on page 103	

wcsncoll

A length limited locale collating wide string comparison.

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

Prototype

```
#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);
```

Parameters	Parameters for this function are:	
	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.

Return 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 103](#)
[“wcsnicoll” on page 116](#)

wcsnicoll

A length limited locale collating wide string case insensitive comparison.

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

Prototype

```
#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);
```

Parameters Parameters for this function are:

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.

Return 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 [“wcsncoll” on page 115](#)
[“strnicoll” on page 107](#)

wcsnicmp

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

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

Prototype

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

```
int _wcsnicmp(const wchar_t *s1,  
              const wchar_t *s2, size_t n);
```

Parameters	Parameters for this function are:	
	s1	const wchar_t * The string being compared
	s2	const wchar_t * The comparison string
	n	int Maximum comparison length

Return The function `_wcsnicmp` 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.

See Also [“strcmp” on page 102](#)
[“strncpy” on page 461](#)

wcsnset

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

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

Prototype

```
#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);
```

Parameters	Parameters for this function are:	
	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.

Return A pointer to the altered string is returned.

See Also [“strset” on page 109](#)

wcsrev

This function reverses a wide character string.

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

Prototype

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

Parameters Parameters for this function are:

str wchar_t * The string to be reversed

Return A pointer to the reversed string.

See Also [“wstrrev” on page 120](#)

wcsset

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

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

Prototype

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

Parameters Parameters for this function are:

str wchar_t * The string to be modified

c wchar_t The char to be set

Return A pointer to the altered string is returned.

See Also [“strnset” on page 108](#)

wcsspnp

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

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

Prototype

```
#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);
```

Parameters Parameters for this function are:

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.

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

See Also ["strspnp" on page 109](#)
["strspn" on page 465](#)

wcsupr

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

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

Prototype

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

Parameters Parameters for this function are:

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

Return A pointer to the converted string.

See Also [“strupr” on page 110](#)

[“strlwr” on page 103](#)

wstrrev

This function reverses a wide character string.

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

Prototype

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

Parameters Parameters for this function are:

str wchar_t * The string to be reversed

Return A pointer to the reversed string.

See Also [“strrev” on page 108](#)

wtoi

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

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

Prototype

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

Parameters Parameters for this function are:

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`.

- Return The function `_wtoi()` returns an the converted integer value.
- See Also [“atoi” on page 406](#)

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” on page 124](#) for creating a file
- [“fcntl” on page 125](#) for file control descriptor
- [“open” on page 126](#) 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” on page 27](#), for information on POSIX naming conventions.

creat

Create a new file or overwrite an existing file.

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

Prototype

```
#include <fcntl.h>
int creat(const char *filename, int mode);
int _creat(const char *filename, int mode);
```

Parameters Parameters for this facility are:

filename	int	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);
```

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

See Also

- [“`open`” on page 316](#)
- [“`fdopen`” on page 305](#)
- [“`close`” on page 499](#).

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.

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

Prototype `#include <fcntl.h>`
`int fcntl(int fildes, int cmd, ...);`
`int _fcntl(int fildes, int cmd, ...);`

Parameters Parameters for this facility are:

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.

Table 14.1 Legal modes with `fcntl()`

Mode	Description
<code>F_DUPFD</code>	Return a duplicate file descriptor.

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

See Also [“fileno” on page 91](#)
[“open” on page 126](#)
[“fdopen” on page 305](#).

Listing 14.2 Example of `fcntl()` usage.

```
#include <unix.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

Opens a file and returns its id.

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

Prototype	#include <fcntl.h> int open(const char *path, int oflag); int _open(const char *path, int oflag);
Parameters	Parameters for this facility are: path char * The file path as a string oflag int The open mode
Remarks	The function <code>open()</code> opens a file for system level input and output. and is used with the UNIX style functions <code>read()</code> and <code>write()</code> .
Table 14.2 Legal file opening modes with <code>open()</code>	
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).
Return	<code>open()</code> returns the file id as an integer value.
See Also	“close” on page 499 “lseek” on page 511

[“read” on page 512](#)

[“write” on page 520](#)

Listing 14.3 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\nnone 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)
{
    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;
```

fcntl.h

Overview of fcntl.h

}

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.

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 131](#), the floating point environment type
- [“`fexcept_t`” on page 131](#), 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 Exception Flags” on page 132,](#)
- [“Rounding Directions” on page 132,](#)
- [“Environment” on page 132.](#)

Floating-Point Exception Flags

<code>FE_DIVBYZERO</code>	Divide by zero
<code>FE_INEXACT</code>	In exact value
<code>FE_INVALID</code>	Invalid value
<code>FE_OVERFLOW</code>	Overflow value
<code>FE_UNDERFLOW</code>	Underflow value
<code>FE_ALL_EXCEPT</code>	The result of a bitwise OR of all the floating-point exception macros

Rounding Directions

<code>FE_DOWNWARD</code>	Rounded downwards
<code>FE_TONEAREST</code>	Rounded to nearest
<code>FE_TOWARDZERO</code>	Rounded to zero
<code>FE_UPWARD</code>	Rounded upwards

Environment

<code>FE_DFL_ENV</code>	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 133,](#) 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.

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

Prototype #pragma STDC FENV_ACCESS on|off|default

Floating-point exceptions

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

- [“feclearexcept” on page 133.](#)
- [“fegetexceptflag” on page 134.](#)
- [“feraiseexcept” on page 136.](#)
- [“fesetexceptflag” on page 136.](#)
- [“fetestexcept” on page 137.](#)

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.

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.

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

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

Parameters Parameters for this function are:

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

Return There is no return value.

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 the argument excepts.

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.

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

Prototype

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

Parameters Parameters for this function are:

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

Return There is no return value.

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 FLAGP */
    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 << "blah\n";
    else
        cout << "success!\n";
}
```

feraiseexcept

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

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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.					
Prototype	#include <fenv.h> void feraiseexcept(int excepts);					
Parameters	Parameters for this function are: <table><tr><td style="padding-right: 20px;">excepts</td><td style="padding-right: 20px;">int</td><td>determines which floating-point exception flags to raise</td></tr></table>			excepts	int	determines which floating-point exception flags to raise
excepts	int	determines which floating-point exception flags to raise				
Return	There is no return value.					

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 argument excepts to the states stored in the object pointed to by the flag.

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

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	<pre>#include <fenv.h> void fetexceptflag(const fexcept_t *flagp, int excepts);</pre>		
Parameters	Parameters for this function are:		
	flag	const fexcept_t *	Constant pointer to floating-point exception flags
	excepts	int	Determines which floating-point exception flags to raise
Return	There is no return value.		

Listing 15.4 Example of fetexceptflag

```
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.

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.

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

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

Parameters Parameters for this function are:

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

Return There is no return value.

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 << "blah\n";
    else
        cout << "success!\n";
}
```

Rounding

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

- [“fegetround” on page 139.](#)

- [“fesetround” on page 140.](#)

fegetround

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

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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <fenv.h> int fegetround(void);
Parameters	This facility has no parameters.
Return	The function fegetround returns the current rounding direction as an integer value.

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.

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.

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

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

Parameters Parameters for this function are:

<code>round</code>	<code>int</code>	Determines the direction result are rounded.
--------------------	------------------	---

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

Listing 15.7 Example of fesetround

Refer to ["Example of fegetround" on page 139,](#)

Environment

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

- ["fegetenv" on page 141,](#)
- ["feholdexcept" on page 141,](#)
- ["fesetenv" on page 143,](#)
- ["feupdateenv" on page 143,](#)

fegetenv

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

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.

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

Prototype `#include <fenv.h>
void fegetenv(fenv_t *envp);`

Parameters Parameters for this function are:

`envp fenv_t *` Pointer to floating-point environment

Return There is no return value.

Listing 15.8 Example of fegetenv

```
long double ie_fegetenv(void)  
{  
    float x=0, y=0;  
    fenv_t env1 =0, env2 = 0;  
    feclearexcept(FE_ALL_EXCEPT);  
  
    fetenv(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) ;
```

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

Prototype `#include <fenv.h>
int feholdexcept(fenv_t *envp)`

Parameters Parameters for this function are:

<code>envp</code>	<code>fenv_t</code>	Pointer to floating-point environment
-------------------	---------------------	---------------------------------------

Return There is no return value.

Listing 15.9 Example of feholdexcept

```
void ie_feholdexcept(void)
{
    /* This function signals underflow if its result is
       denormalized, overerflow 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`.

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

Prototype `#include <fenv.h>
void fesetenv(const fenv_t *envp);`

Parameters Parameters for this function are:

`envp const fenv_t` Pointer to floating-point environment

Return There is no return value.

Listing 15.10 Example of fesetenv

Refer to ["Example of fegetenv" on page 141.](#)

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.

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

Prototype `#include <fenv.h>`
 `void feupdateenv(const fenv_t *envp);`

Parameters Parameters for this function are:

`envp` `const fenv_t` Constant pointer to floating-point
 environment

Return There is no return value.

Listing 15.11 Example of feupdateenv

Refer to ["Example of feholdexcept" on page 142.](#)

float.h

The float.h header file macros specify the “[Floating point number characteristics](#)” on page 145 for float, double and long double types.

Overview of float.h

Floating point number characteristics

The float.h header file consists of macros that specify the characteristics of floating point number representation for float, double and long double types.

- These macros are listed in the listing “[Floating point characteristics](#)” on page 145

“[Floating point characteristics](#)” on page 145 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.

Table 16.1 Floating point characteristics

Macro	Description
FLT_MANT_DIG, DBL_MANT_DIG, LDBL_MANT_DIG	The number of base FLT_RADIX digits in the significant.

FLT_DIG, DBL_DIG, LDBL_DIG	The decimal digit precision.
FLT_MIN_EXP, DBL_MIN_EXP, LDBL_MIN_EXP	The smallest negative integer exponent that <code>FLT_RADIX</code> can be raised to and still be expressible.
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 <code>FLT_RADIX</code> 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 147](#) a Macintosh file opening for fopen
- [“FSRef_fopen” on page 148](#) a Macintosh file opening for fopen when files exist.
- [“FSRefParentAndFilename_fopen” on page 148](#) 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.

Compatibility	Macintosh only, this function may not be implemented on all Mac OS versions.							
Prototype	<pre>#include <Fsp_fopen.h> FILE * FSp_fopen (ConstFSSpecPtr spec, const char * open_mode);</pre>							
Parameters	Parameters for this facility are:							
	<table> <tr> <td>spec</td> <td>ConstFSSpecPtr</td> <td>A toolbox file pointer</td> </tr> <tr> <td>open_mode</td> <td>char *</td> <td>The open mode</td> </tr> </table>		spec	ConstFSSpecPtr	A toolbox file pointer	open_mode	char *	The 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.							

FSp_fopen.h

Overview of FSp_fopen.h

Return The FSp_fopen facility opens a file with the Macintosh Toolbox FSRefPtr function and return a FILE pointer.

See Also [“fopen” on page 316](#)

FSRef_fopen

Opens an existing file with FSRef and return a FILE pointer.

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

Prototype

```
#include <FSp_fopen.h>
FILE * FSRef_fopen (FSRefPtr spec,
                    const char * open_mode);
```

Parameters Parameters for this facility are:

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

Return The FSRefPtr facility returns a FILE pointer

See Also [“fopen” on page 316](#)
[“FSRefParentAndFilename_fopen” on page 148](#)

FSRefParentAndFilename_fopen

FSRefParentAndFilename_fopen works on both files that already exist as well as nonexistent files which can be created by the call.

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

Prototype	<pre>#include <Fsp_fopen.h> FILE * FSRefParentAndFilename_fopen(const FSRefPtr theParentRef, ConstHFSUniStr255Param theName, const char *open_mode);</pre>	
Parameters	Parameters for this facility are:	
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	<p>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.</p> <ul style="list-style-type: none"> • TextCommon • UnicodeConverter • UTCUtils 	
Return	The FSRefParentAndFilename_fopen facility returns a FILE pointer	
See Also	“fopen” on page 316 “FSRef fopen” on page 148	

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 152](#) a type used for long long division
- [“Greatest-Width Format Specifier Macros” on page 152](#) formatting specifiers for printf and scanf family functions.
- [“Greatest-Width Integer Functions” on page 155](#) functions for manipulation and conversion

The header `inttype.h` includes several functions for greatest-width integer manipulation and conversions.

- [“imaxabs” on page 155](#) computes the absolute value of a greatest-width integer.
- [“imaxdiv” on page 155](#) computes the quotient and remainder
- [“strtoimax” on page 156](#) string to greatest-width integer
- [“strtoumax” on page 157](#) string to greatest-width unsigned integer
- [“wcstoiymax” on page 158](#) wide character string to greatest-width integer
- [“wcstoumax” on page 159](#) 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 152](#).

Table 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 Format Specifier 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 152](#) describes output formatting macros

The table [“Fscanf Greatest-Width Format Specifiers” on page 154](#) describes input formatting macros.

Table 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	PRIi8	i

Macro Substitution	Specifier	Macro Substitution	Specifier	Macro Substitution	Specifier
PRIi16	hi	PRIi32	li	PRIi64	lli
PRIiLEAST8	i	PRIiLEAST16	hi	PRIiLEAST32	li
PRIiLEAST64	lli	PRIiFAST8	i	PRIiFAST16	hi
PRIiFAST32	li	PRIiFAST64	lli	PRIiMAX	lli
PRIiPTR	li	PRIo8	o	PRIo16	ho
PRIo32	lo	PRIo64	llo	PRIoLEAST8	o
PRIoLEAST16	ho	PRIoLEAST32	lo	PRIoLEAST64	llo
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	lX	PRIX64	llX	PRIXLEAST8	X
PRIXLEAST16	hX	PRIXLEAST32	lX	PRIXLEAST64	llx
PRIXFAST8	X	PRIXFAST16	hX	PRIXFAST32	lX
PRIXFAST64	llX	PRIXMAX	llX	PRIXPTR	lX

NOTE Separate macros are used with input and output functions because different format specifiers are generally required for the `fprintf` and `scanf` family of functions.

Table 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
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 `inttype.h` includes several functions for greatest-width integer manipulation and conversions.

imaxabs

Computes the absolute value of a greatest-width integer.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype `#include <inttypes.h>`
 `intmax_t imaxabs(intmax_t j);`

Parameters Parameters for this facility are:

`j` `intmax_t` The value being computed

Remarks The behavior is undefined if the result can not be represented.

Return The `imaxabs` function returns the absolute value.

See Also [“abs” on page 402](#)

[“labs” on page 418](#)

imaxdiv

Compute the greatest-width integer quotient and remainder.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype `#include <inttypes.h>`
 `imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);`

Parameters Parameters for this facility are:

`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.

Return	The <code>imaxdiv</code> function returns a structure of type “imaxdiv_t” on page 152 storing both the quotient and the remainder values.
See Also	“div” on page 414 “ldiv” on page 419

strtoimax

Character array conversion to a greatest-width integral value.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype

```
#include <inttypes.h>
intmax_t strtointmax(const char * restrict nptr,
                      char ** restrict endptr, int base);
```

Parameters Parameters for this facility are:

<code>nptr</code>	<code>const char *</code>	A character array to convert
<code>endptr</code>	<code>char **</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 `strtointmax()` function converts a character array, pointed to by `nptr`, expected to represent an integer expressed in 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.

Return 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`.

See Also [“strtoumax” on page 157](#)
[“strtol” on page 431](#)
[“strtoll” on page 435](#)

strtoumax

Character array conversion to a greatest-width integer value.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype

```
#include <inttypes.h>
uintmax_t strtoumax(const char * restrict nptr,
                     char ** restrict endptr, int base);
```

Parameters Parameters for this facility are:

<code>nptr</code>	<code>const char *</code>	A character array to convert
<code>endptr</code>	<code>char **</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 `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.

Return 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`.

See Also [“strtoimax” on page 156](#)

[“strtoul” on page 436](#)

[“strtoull” on page 438](#)

wcstoimax

Wide character array conversion to a greatest-width integral value.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype

```
#include <inttypes.h>
intmax_t wcstoimax(const wchar_t * restrict nptr,
                    wchar_t ** restrict endptr, int base);
```

Parameters Parameters for this facility are:

<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 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 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.

Return 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.

See Also [“wcstoumax” on page 159](#)

wcstoumax

Wide character array conversion to a greatest-width integer value.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype

```
#include <inttypes.h>
uintmax_t wcstoumax(const wchar_t * restrict nptr,
                     wchar_t ** restrict endptr, int base);
```

Parameters Parameters for this facility are:

nptr	const wchar_t *	A 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 wcstoumax() 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 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 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.

Return 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`.

See Also [“wcstoiimax” on page 158](#)

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 162](#) closes a directory search
- [“_findfirst” on page 162](#) opens a directory search
- [“_findnext” on page 163](#) searches a directory

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” on page 27](#), 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`.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype

```
#include <io.h>
int _findclose(long fhandle);
```

Parameters Parameters for this facility are:

`fhandle` long The size in bytes of the allocation

Remarks The `fhandle` is returned by `_findfirst`.

Return Zero is returned on success and a negative one on failure.

See Also [“`findfirst`” on page 162](#)

[“`findnext`” on page 163](#)

_findfirst

Searches a directory folder.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype

```
#include <io.h>
long _findfirst(const char *pathname,
                 struct _finddata_t * fdata);
```

Parameters Parameters for this facility are:

`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.

Return A file handle is returned.

See Also [“`finddata_t`” on page 161](#)

[“`findnext`” on page 163](#)

[“ `findclose`” on page 162](#)

_findnext

Continues a directory search began with `_findfirst`.

Compatibility Windows compatible header yet may also be implemented in other headers.

Prototype

```
#include <io.h>
int _findnext(long fhandle,
              struct _finddata_t * fdata);
```

Parameters Parameters for this facility are:

<code>fhandle</code>	<code>long</code>	The handle returned from calling <code>_findfirst</code>
<code>fdata</code>	<code>_finddata_t *</code>	A pointer to a file directory information structure

Remarks Wildcards are allowed in the directory search.

Return Zero is returned if the file is found, if no file is found then a negative one is returned.

See Also

- [“ `finddata_t`” on page 161](#)
- [“ `findfirst`” on page 162](#)
- [“ `findclose`” on page 162](#)

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 165](#).

Table 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 167.](#)

Integral type limits

The `limits.h` header file macros describe the maximum and minimum values of integral types.

[“Integral limits” on page 167](#) describes the macros.

Table 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> .

SHRT_MAX	Maximum value for an object of type <code>short int</code> .
SHRT_MIN	Minimum value for an object of type <code>short int</code> .
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 169 are:

- “[lconv structure and contents returned by `localeconv\(\)`](#)” on page 169
- “[`localeconv`](#)” on page 170 to get the locale
- “[`setlocale`](#)” on page 171 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 169](#). An `lconv` member is assigned “[CHAR_MAX](#)” on page 167 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.

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

Prototype `#include <locale.h>`
`struct lconv *localeconv(void);`

Parameters None

Return `localeconv()` returns a pointer to an `lconv` structure for the "C" locale. Refer to Figure 1.

setlocale

Query or set locale information for the C compiler.

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

Prototype

```
#include <locale.h>
char *setlocale(
    int category,
    const char *locale);
```

Parameters Parameters for this facility are:

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 or an empty string, 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" locale.

Attempting to set a part of the Metrowerks C compiler's locale will have no effect.

See Also ["strcoll" on page 454](#)

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 173](#) 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” on page 27](#), for information on POSIX naming conventions.

alloc

Allocates memory quickly on the stack.

Compatibility Windows compatible yet may also be implemented in other headers.

Prototype `#include <malloc.h>`
`void *alloc(size_t nbytes);`

Parameters Parameters for this facility are:

`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.

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

See Also [“calloc” on page 412](#)
[“free” on page 416](#)
[“malloc” on page 421](#)
[“realloc” on page 427](#)

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 94](#), 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 180](#) classifies floating point numbers
- [“isfinite” on page 181](#) tests if a value is a finite number
- [“isnan” on page 181](#) test if a value is a computable number
- [“isnormal” on page 181](#) tests for normal numbers
- [“signbit” on page 182](#) tests for a negative number

Functions

- [“acos” on page 182](#) determines the arccosine
- [“asin” on page 184](#) determines the arcsine
- [“atan” on page 185](#) determines the arctangent
- [“atan2” on page 186](#) determines the arctangent of two variables
- [“ceil” on page 187](#) determines the smallest int not less than x
- [“cos” on page 188](#) determines the cosine
- [“cosh” on page 189](#) determines the hyperbolic cosine
- [“exp” on page 190](#) computes the exponential
- [“fabs” on page 192](#) determines the absolute value
- [“floor” on page 193](#) determines the largest integer not greater than x
- [“fmod” on page 194](#) determines the remainder of a division

- [“`frexp`” on page 196](#) extracts a value of the mantissa and exponent
- [“`ldexp`” on page 199](#) computes a value from a mantissa and exponent
- [“`log`” on page 200](#) determines the natural logarithm
- [“`log10`” on page 202](#) determines the logarithm to base 10
- [“`modf`” on page 203](#) separates integer and fractional parts
- [“`pow`” on page 204](#) raises to a power
- [“`sin`” on page 206](#) determines the sine
- [“`sinh`” on page 207](#) determines the hyperbolic sine
- [“`sqrt`” on page 208](#) determines the square root
- [“`tan`” on page 209](#) determines the tangent
- [“`tanh`” on page 211](#) determines the hyperbolic tangent

C9X Implementations

- [“`acosh`” on page 212](#) computes the (non-negative) arc hyperbolic cosine
- [“`asinh`” on page 213](#) computes the arc hyperbolic sine
- [“`atanh`” on page 214](#) computes the arc hyperbolic tangent
- [“`copysign`” on page 215](#) produces a value with the magnitude of x and the sign of y
- [“`erf`” on page 216](#) computes the error function
- [“`erfc`” on page 216](#) complementary error function
- [“`exp2`” on page 217](#) computes the base-2 exponential
- [“`expm1`” on page 218](#) Computes the exponential minus 1
- [“`fdim`” on page 219](#) computes the positive difference of its arguments
- [“`fmax`” on page 219](#) computes the maximum numeric value of its argument
- [“`fmin`” on page 220](#) computes the minimum numeric value of its arguments
- [“`gamma`” on page 221](#) computes the gamma function
- [“`hypot`” on page 222](#) computes the square root of the sum of the squares of the arguments

- [“`isgreater`” on page 197](#) compares two numbers for x greater than y
- [“`isgreaterless`” on page 197](#) compares numbers for x not equal to y
- [“`isless`” on page 198](#) compares two numbers for x less than y
- [“`islessequal`” on page 198](#) compares two numbers for x is less than or equal to y
- [“`isunordered`” on page 199](#) compares two numbers for lack of order
- [“`lgamma`” on page 223](#) computes the log of the absolute value
- [“`log1p`” on page 223](#) computes the natural- log of x plus 1
- [“`log2`” on page 224](#) computes the base-2 logarithm
- [“`logb`” on page 225](#) extracts the exponent of a double value
- [“`nan`” on page 226](#) Tests for NaN
- [“`nearbyint`” on page 227](#) rounds off the argument to an integral value
- [“`nextafter`” on page 227](#) determines the next representable value in the type of the function
- [“`remainder`” on page 229](#) computes the remainder x REM y required by IEC 559
- [“`remquo`” on page 230](#) computes the same remainder as the remainder function
- [“`rint`” on page 230](#) rounds off the argument to an integral value
- [“`rinttol`” on page 231](#) rinttol rounds its argument to the nearest long integral value
- [“`round`” on page 232](#) rounds its argument to an integral value in floating-point format
- [“`roundtol`” on page 232](#) roundtol rounds its argument to the nearest integral value
- [“`scalb`” on page 233](#) computes $x * \text{FLT_RADIX}^n$
- [“`trunc`” on page 233](#) 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 “[strtod](#)” on page 431 for information on `strtod()`.

Un-optimized x86 `math.h` functions may use the “[errno](#)” global variable to indicate an error condition. In particular, many functions set `errno` to `EDOM` (see [Table 12.1 on page 84](#)) 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 180](#), 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

See Also [“NaN Not a Number” on page 178](#)

fpclassify

Classifies floating point numbers.

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

Prototype `#include <math.h>
int __fpclassify(long double x);
int __fpclassifyd(double x);
int __fpclassifyf(float x);`

Parameters Parameters for this facility are:

x float, double or long double number evaluated

Return An integral value FP_NAN, FP_INFINITE, FP_ZERO, FP_NORMAL and FP_SUBNORMAL.

See Also [“isfinite” on page 181](#)

[“isnan” on page 181](#)

[“isnormal” on page 181](#)

[“signbit” on page 182](#)

[“NaN Not a Number” on page 178](#)

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.

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

Prototype `#include <math.h>`
`int isfinite(double x);`

Parameters Parameters for this facility are:

`x` float, double or long double number evaluated

Return The facility returns true if the value tested is finite otherwise it returns false.

See Also [“fpclassify” on page 180](#)

isnan

The facility `isnan` test if a value is a computable number.

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

Prototype `#include <math.h>`
`int isnan (double x);`

Parameters Parameters for this facility are:

`x` float, double or long double number evaluated

Return This facility is true if the argument is not a number.

See Also [“fpclassify” on page 180](#)
[“NaN Not a Number” on page 178](#)

isnormal

A test of a normal number.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <math.h> int isnormal(double x);
Parameters	Parameters for this facility are:
	x float, double or long double number evaluated
Return	This facility is true if the argument is a normal number.
See Also	“fpclassify” on page 180

signbit

A test for a number that includes a signed bit

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <math.h> int __signbit(long double x); int __signbitd(double x); int __signbit(float x);
Parameters	Parameters for this facility are:
	x float, double or long double number evaluated
Return	This facility is true if the sign of the argument value is negative.
See Also	“fpclassify” on page 180

Floating Point Math Facilities

acos

This function computes the arc values of cosine, sine, and tangent.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <math.h> double acos(double x); float acosf(float); long double acosl(long double);
Parameters	Parameters for this function are: x float, double or long double value to be computed
Return	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 179 , for information on newer error testing procedures.
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 179 , for information on newer error testing procedures. See “Example of acos(), asin(), atan(), atan2() usage.” on page 186 for example usage.
See Also	“Inlined Intrinsics Option” on page 179 “cos” on page 188 “errno” on page 83

acosf

Implements the acos() function for float type values. See [“acos” on page 182](#).

acosl

Implements the acos() function for long double type values. See [“acos” on page 182](#).

asin

Arcsine function.

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

Prototype

```
#include <math.h>

double asin(double x);
float asinf(float);
long double asinl(long double);
```

Parameters Parameters for this function are:

x float, double or long double value to be computed

Return 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 179](#) for information on newer error testing procedures.

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 179](#) for information on newer error testing procedures.

See “[Example of acos\(\), asin\(\), atan\(\), atan2\(\) usage.” on page 186](#) for example usage.

See Also [“Inlined Intrinsics Option” on page 179](#)

[“sin” on page 206](#)

[“errno” on page 83](#)

asinf

Implements the `asin()` function for float type values. See “[asin” on page 184](#).

asinl

Implements the asin() function for long double type values. See [“asin” on page 184.](#)

atan

Arctangent function. This function computes the value of the arc tangent of the argument.

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

Prototype

```
#include <math.h>

double atan(double x);
float atanf(float);
long double atanl(long double);
```

Parameters Parameters for this function are:

x float, double or long double value to be computed

Return 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 186](#) for example usage.

See Also [“tan” on page 209](#)
[“errno” on page 83](#)

atanf

Implements the atan() function for float type values. See [“atan” on page 185.](#)

atanl

Implements the atan() function for long double type values. See [“atan” on page 185.](#)

atan2

Arctangent function. This function computes the value of the tangent of x/y using the sines of both arguments.

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

Prototype

```
#include <math.h>

double atan2(double y, double x);
float atan2f(float, float);
long double atan2l(long double, long double);
```

Parameters Parameters for this function are:

y	double, float or long double	Value one
x	double, float or long double	Value two

A domain error occurs if both x and y are zero.

Return 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 186 for example usage.

See Also [“Inlined Intrinsics Option” on page 179](#)
[“tan” on page 209](#)
[“errno” on page 83](#)

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 186](#).

atan2l

Implements the atan2() function for long double type values. See [“atan2” on page 186](#).

ceil

Compute the smallest floating point number not less than x.

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

Prototype `#include <math.h>`

`double ceil(double x);`
`float ceilf(float);`
`long double ceill(long double);`

Parameters Parameters for this function are:

`x` float, double or long double value to be computed

Return `ceil()` returns the smallest integer not less than x.

See Also [“floor” on page 193](#)
[“fmod” on page 194](#)

[“round” on page 232](#)

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 187.](#)

ceil

Implements the ceil() function for long double type values. See [“ceil” on page 187.](#)

cos

Compute cosine.

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

Prototype

```
#include <math.h>

double cos(double x);
float cosf(float);
long double cosl(long double);
```

Parameters	Parameters for this function are:	
	x	float, double or long double value to be computed
Return	cos () returns the cosine of x. x is measured in radians.	
See Also	“sin” on page 206 “tan” on page 209	

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 188](#).

cosl

Implements the cos() function for long double type values. See [“cos” on page 188](#).

cosh

Compute the hyperbolic cosine.

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

Prototype `double cosh(double x);`

```
float coshf(float);
long double coshl(long double);
```

Parameters Parameters for this function are:

 x float, double or long double value to be computed

Return `cosh()` returns the hyperbolic cosine of x.

See Also [“Inlined Intrinsics Option” on page 179](#)

[“sinh” on page 207](#)

[“tanh” on page 211](#)

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 189.](#)

coshl

Implements the `cosh()` function for long double type values. See [“cosh” on page 189.](#)

exp

Computes the exponent of the function’s argument

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

Prototype `#include <math.h>`

```
double exp(double x);
float expf(float);
long double expl(long double);
```

Parameters Parameters for this function are:

x float, double or long double value to be computed

Return `exp()` returns e^x , where e is the natural logarithm base value.

Remarks A range error may occur for larger numbers.

See Also [“Inlined Intrinsics Option” on page 179](#)
[“log” on page 200](#)
[“expm1” on page 218](#)
[“exp2” on page 217](#)
[“pow” on page 204](#)

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 190.](#)

expl

Implements the exp() function for long double type values. See [“exp” on page 190.](#)

fabs

Compute the floating point absolute value.

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

Prototype

```
#include <math.h>

double fabs(double x);
float fabsf(float);
long double fabsl(long double);
```

Parameters Parameters for this function are:

x	float, double or long double	value to be computed
---	------------------------------	----------------------

Return `fabs()` returns the absolute value of `x`.

See Also [“floor” on page 193](#)

[“ceil” on page 187](#)

[“fmod” on page 194](#)

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 192](#).

fabsl

Implements the fabs() function for long double type values. See [“fabs” on page 192](#).

floor

Compute the largest floating point not greater than x .

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

Prototype `#include <math.h>`

`double floor(double x);`
`float floorf(float);`
`long double floorl(long double);`

Parameters Parameters for this function are:

x float, double or long double value to be computed

Return The function `floor()` returns the largest integer not greater than x .

See Also [“ceil” on page 187](#)
[“fmod” on page 194](#)
[“fabs” on page 192](#)

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 193.](#)

floorl

Implements the floor() function for long double type values. See [“floor” on page 193.](#)

fmod

Return the floating point remainder of x / y .

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

Prototype `#include <math.h>`

```
double fmod(double x, double y);
float fmodf(float, float);
long double fmodl(long double, long double);
```

Parameters Parameters for this function are:

x double, float or long double The value to compute

y double, float or long double The divider

Return 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 .

See Also [“floor” on page 193](#)
[“ceil” on page 187](#)
[“fmod” on page 194](#)
[“fabs” on page 192](#)

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 194](#).

fmodl

Implements the fmod() function for long double type values. See [“fmod” on page 194](#).

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.

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

Prototype

```
#include <math.h>

double frexp(double value, int *exp);
float frexpf(float, int *);
long double frexpl(long double, int *);
```

Parameters Parameters for this function are:

x	double, float or long double	The value to compute
exp	int	Exponent

Return `frexp()` returns the double mantissa of value. It stores the integer exponent value at the address referenced by `exp`.

See Also [“ldexp” on page 199](#)
[“fmod” on page 194](#)

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 [“frexpf” on page 196](#).

frexpl

Implements the frexp() function for long double type values. See [“frexpl” on page 196](#).

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.

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

Prototype
`#include <math.h>`
`int isgreater(x, y)`

Parameters Parameters for this facility are:

<code>x</code>	float, double or long double	number compared
<code>y</code>	float, double or long double	number compared

Return This facility is true if `x` is greater than `y`.

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.

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

Prototype `int isgreaterless(x, y)`

Parameters Parameters for this facility are:

`x` float, double or long double number compared

`y` float, double or long double number compared

Return This facility returns true if `x` is greater than or less than `y`.

isless

The facility determines the lesser of two numbers.

Compatibility This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list. Unlike `x < y` `isless` does not raise an invalid exception when `x` and `y` are unordered.

Prototype `#include <math.h>`

`int isless(x, y)`

Parameters Parameters for this facility are:

`x` float, double or long double number compared

`y` float, double or long double number compared

Return This facility is true if `x` is less than `y`.

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.

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

Prototype `#include <math.h>`
 `int islessequal(x, y)`

Parameters Parameters for this facility are:

x	float, double or long double	number compared
y	float, double or long double	number compared

Return This facility is true if x is less than or equal to y.

isunordered

The facility compares the order of the arguments.

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

Prototype `int isunordered(x, y)`

Parameters Parameters for this facility are:

x	float, double or long double	number compared
y	float, double or long double	number compared

Return This facility is true if the arguments are unordered false otherwise.

ldexp

Compute a value from a mantissa and exponent. The `ldexp()` function computes $x * 2^{\text{exp}}$. This function can be used to construct a double value from the values returned by the `frexp()` function.

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

Prototype
`#include <math.h>`

`double ldexp(double x, int exp);`
`float ldexpf(float, int);`
`long double ldexpl(long double, int);`

Parameters Parameters for this function are:

x double, float or long double The value to compute

exp int Exponent

Return ldexp() returns $x * 2^{\text{exp}}$.

See Also [“frexp” on page 196](#)

[“modf” on page 203](#)

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 199](#).

ldexpl

Implements the ldexp() function for long double type values. See [“ldexp” on page 199](#).

log

Compute the natural logarithms.

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

Prototype

```
#include <math.h>

double log(double x);
float logf(float);
long double logl(long double);
```

Parameters Parameters for this function are:

x float, double or long double value to be computed

Return log() returns log_ex. If x < 0 the log() may assign EDOM to errno

See “[Floating point error testing.](#)” on page 179, for information on newer error testing procedures.

See Also [“Inlined Intrinsics Option” on page 179](#)

[“exp” on page 190](#)

[“errno” on page 83](#)

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 200.](#)

logl

Implements the log() function for long double type values. See [“log” on page 200.](#)

log10

Compute the base 10 logarithms.

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

Prototype

```
#include <math.h>

double log10(double x);
float log10f(float);
long double log10l(long double);
```

Parameters Parameters for this function are:

x float, double or long double value to be computed

Return $\log_{10}x$. If $x < 0$ $\log_{10}()$ may assign EDOM to errno. See [“Floating point error testing.” on page 179](#), for information on newer error testing procedures.

See Also [“Inlined Intrinsics Option” on page 179](#)
[“exp” on page 190](#)
[“errno” on page 83](#)

Listing 24.13 For example of usage see:

[“log\(\), log10\(\) example” on page 201](#)

log10f

Implements the log10() function for float type values. See “[log10](#)” on page 202.

log10l

Implements the log10() function for long double type values. See “[log10](#)” on page 202.

modf

Separate integer and fractional parts.

The modf() function separates value into its integer and fractional parts. In other words, modf() separates value such that value = $f + i$ where $0 \leq f < 1$, and i is the largest integer that is not greater than value.

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

Prototype

```
#include <math.h>

double modf(double value, double *iptr);
float fmodf(float value, float *iptr);
long double modfl(long double value,
                  long double *iptr);
```

Parameters Parameters for this function are:

value	double, float, or long double	The value to separate
iptr	double, float, or long double	integer part

Return modf() returns the signed fractional part of value, and stores the integer part in the integer pointed to by iptr.

See Also [“frexp” on page 196](#)
[“ldexp” on page 199](#)

Listing 24.14 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 203.](#)

modfl

Implements the modf() function for long double type values. See [“modf” on page 203.](#)

pow

Calculate x^y .

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

Prototype

```
#include <math.h>

double pow(double x, double y);
float powf(float, float x);
long double powl(long double, long double x);
```

Parameters	Parameters for this function are:	
	x	float, double or long double value to be computed
Return	pow() returns x^y .	
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 Also	“Floating point error testing.” on page 179 , for information on newer error testing procedures. “Inlined Intrinsics Option” on page 179 “sqrt” on page 208 “Example usage of error detection” on page 180.	

Listing 24.15 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 204.](#)

powl

Implements the pow() function for long double type values. See [“pow” on page 204.](#)

sin

Compute sine.

Compatibility This function may not be implemented on all platforms. Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list. The argument for the sin() function should be in radians. One radian is equal to $360/2\frac{1}{4}$ degrees.

Prototype `#include <math.h>`

```
double sin(double x);
float sinf(float x);
long double sinl(long double x);
```

Parameters Parameters for this function are:

`x` float, double or long double value to be computed

Return `sin()` returns the sine of `x`. `x` is measured in radians.

See Also [“cos” on page 188](#)
[“tan” on page 209](#)

Listing 24.16 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 206](#).

sinl

Implements the sin() function for long double type values. See [“sin” on page 206](#).

sinh

Compute the hyperbolic sine.

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

Prototype `#include <math.h>`

```
double sinh(double x);
float sinhf(float x);
long double sinhl(long double x);
```

Parameters Parameters for this function are:

`x` float, double or long double value to be computed

Return `sinh()` returns the hyperbolic sine of `x`.

Remarks A range error can occur if the absolute value of the argument is too large.

See Also [“Inlined Intrinsics Option” on page 179](#)

[“cosh” on page 189](#)

[“tanh” on page 211](#)

Listing 24.17 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 207.](#)

sinhl

Implements the sinh() function for long double type values. See [“sinh” on page 207.](#)

sqrt

Calculate the square root.

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

Prototype

```
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

Parameters Parameters for this function are:

	x	float, double or long double	value to compute
Return		sqrt()	returns the square root of x.
Remarks		A domain error occurs if the argument is a negative value.	
See Also		“Inlined Intrinsics Option” on page 179	“pow” on page 204

Listing 24.18 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 208.](#)

sqrtl

Implements the sqrt() function for long double type values. See [“sqrt” on page 208.](#)

tan

Computes tangent of the argument.

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

Prototype `#include <math.h>`

```
double tan(double x);
float tanf(float x);
long double tanl(long double x);
```

Parameters Parameters for this function are:

<code>x</code>	float, double or long double	value to compute
----------------	------------------------------	------------------

Return `tan()` returns the tangent of `x`. `x` is measured in radians.

Remarks A range error may occur if the argument is close to an odd multiple of pi divided by 2

See Also [“Inlined Intrinsics Option” on page 179](#)

[“cos” on page 188](#)

[“sin” on page 206](#)

Listing 24.19 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 209.](#)

tanl

Implements the tan() function for long double type values. See [“tan” on page 209.](#)

tanh

Compute the hyperbolic tangent.

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

Prototype `#include <math.h>`

`double tanh(double x);`
`float tanhf(float x);`
`long double tanhl(long double x);`

Parameters Parameters for this function are:

`x` float, double or long double value to compute

Return `tanh()` returns the hyperbolic tangent of `x`.

See Also [“cosh” on page 189](#)
[“sinh” on page 207](#)

Listing 24.20 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 211](#).

tanhl

Implements the tanh() function for long double type values. See “[tanh](#)” on [page 211](#).

HUGE_VAL

The largest floating point value with the same sign possible for a function’s return.

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

Prototype `#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.

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, +INF].

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

Prototype	#include <math.h> double acosh (double x);	
Parameters	Parameters for this function are:	
	x	double The value to compute
Return	The (non-negative) arc hyperbolic cosine of x.	
Remarks	A domain error occurs for argument x is less than 1 and a range error occurs if x is too large.	
See Also	“acos” on page 182	

Listing 24.21 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.

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

Prototype #include <math.h>
double asinh (double x);

Parameters Parameters for this function are:

x double The value to compute

Return The arc hyperbolic sine of the argument x.

Remarks A range error occurs if the magnitude of x is too large.

See Also [“asin” on page 184](#)

Listing 24.22 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.

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

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

Parameters Parameters for this function are:

`x` double The value to compute

Return The arc hyperbolic tangent of `x`.

Remarks A domain error occurs for arguments not in the range [-1,+1]

See Also [“atan” on page 185](#)

Listing 24.23 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.
```

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.

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

Prototype `#include <math.h>`
`double copysign (double x, double y);`

Parameters Parameters for this function are:

<code>x</code>	double	Magnitude
<code>y</code>	double	The sign argument

Return A value with the magnitude of `x` and the sign of `y`.

Listing 24.24 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. erf() is defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

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

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

Parameters Parameters for this function are:

`x` double The value to be computed

Return The error function of `x`.

Listing 24.25 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 of `x`:

$$\text{erfc}(x) = 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$$

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

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

Parameters Parameters for this function are:
 `x` double The value to be computed

Return The complementary error function of `x`.

Listing 24.26 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. In other words `exp2(b)` solves for the `x` in $(b = 2^x)$.

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

Prototype `#include <math.h>
double exp2 (double x);`

Parameters Parameters for this function are:
 `x` double The value to compute

Remarks A range error occurs if the magnitude of `x` is too large

Return The function returns the base-2 exponential of `x`: 2^x

See Also [“pow” on page 204](#)

Listing 24.27 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.
Written as:

$$\text{expm1}(x) = (\text{ex}) - 1.0$$

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

Prototype `#include <math.h>`
`double expm1 (double x);`

Parameters Parameters for this function are:

`x` double The value to compute

Return The base-e exponential of `x`, minus 1: $(e^x) - 1$.

Remarks A range error occurs if `x` is too large. For small magnitude `x`, `expm1(x)` is expected to be more accurate than `exp(x) - 1`

Listing 24.28 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

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

Prototype `#include <math.h>`
`double fdim (double x, double y);`

Parameters Parameters for this function are:

x	double	Value one
y	double	Value two

Return 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

Listing 24.29 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.
```

fmax

The function **fmax** computes the maximum numeric value of its argument

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	#include <math.h> double fmax (double x, double y);		
Parameters	Parameters for this function are:		
	x	double	First argument
	y	double	Second argument
Return	The maximum value of x or y.		
See Also	“fmin” on page 220		

Listing 24.30 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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	#include <math.h> double fmin (double x, double y);		
Parameters	Parameters for this function are:		

	x	double	First argument
	y	double	Second argument
Return	Fmin returns the minimum numeric value of its arguments		
See Also	“fmax” on page 219		

Listing 24.31 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 $\log_e G(x)$, 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).

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

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

Parameters Parameters for this function are:

x double The value to be computed

Return The gamma function of `x`.

Remarks	An application wishing to check for error situations should set <code>errno</code> to 0 before calling <code>lgamma()</code> . If <code>errno</code> is non-zero on return. <ul style="list-style-type: none">• If the return value is NaN, an error has occurred.• A domain error occurs if <code>x</code> is equal to zero or if <code>x</code> is a negative integer.• A range error may occur.
See Also	“lgamma” on page 223

hypot

The function `hypot` computes the square root of the sum of the squares of the arguments. Hypot computes the square root of the sum of the squares of `x` and `y` without undue overflow or underflow.

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

Prototype `#include <math.h>`
 `double hypot (double x, double y);`

Parameters	Parameters for this function are:
	<code>x</code> double The first value to be squared
	<code>y</code> double The second value to be squared

Return The square root of the sum of the squares of `x` and `y`.

Remarks A range error may occur.

See Also [“Inlined Intrinsics Option” on page 179](#)

Listing 24.32 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
```

Igamma

The `lgamma()` function computes the same thing as the `gamma()` with the addition of absolute value signs $\log_e |G(x)|$, where $G(x)$ is defined as the integral of $(e^{-t} * t^{(x-1)})dt$ from 0 to infinity.

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

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

Parameters Parameters for this function are:

x double The value to be computed

Remarks 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

Return The log of the absolute value of gamma of x .

See Also [“gamma” on page 221](#)

log1p

The function `log1p` computes the base-e logarithm. Which can be denoted as

log1p = loge(1.0 + x)

The value of x must be greater than -1.0.

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

Prototype `#include <math.h>
double log1p (double x);`

Parameters Parameters for this function are:

x double The value being computed

Return The base-e logarithm of 1 plus x.

Remarks 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.

See Also [“log” on page 200](#)

Listing 24.33 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 which can be denoted as:

`log2(x) = log2(x)`

The value of `x` must be positive.

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

Prototype `#include <math.h>`
`double log2 (double x);`

Parameters Parameters for this function are:

`x` double The value being computed

Return The base-2 logarithm of `x`.

Remarks • A domain error may occur if `x` is less than zero.
• A range error may occur if `x` is equal to zero.

See Also [“log” on page 200](#)

Listing 24.34 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`, 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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <math.h> double logb (double x);	
Parameters	Parameters for this function are:	
	x	double The value being computed
Remarks	A range error may occur if x is equal to zero.	
Return	The exponent of x as a signed integral value in the format of the x argument.	
See Also	“Inlined Intrinsics Option” on page 179	

Listing 24.35 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

nan

The function nan tests for NaN.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <math.h> double nan(const char *tagp);	
Parameters	Parameters for this function are:	
	tagp	const char * A character string

Return	A quiet NaN if available.
	See “ Quiet NaN ” on page 178, fore more information.
See Also	isnan on page 181 “NaN Not a Number” on page 178

nearbyint

The function `nearbyint` rounds off the argument to an integral value.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <math.h> double nearbyint (double x);
Parameters	Parameters for this function are: x double The value to be computed
Return	The argument in integral value in floating point format.
Remarks	Nearbyint, computes like <code>rint</code> but doesn't raise an inexact exception.

Listing 24.36 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, 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 .

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

Defined

```
#include <math.h>

#define nextafter(x,y)
    ( (sizeof(x) == sizeof(float)) ?
        nextafterf(x,y) :
        (sizeof(x) == sizeof(double)) ?
            nextafterd(x,y)
```

Parameters Parameters for this macro are:

x	float double long double	current representable value
y	float double long double	direction

Return The next representable value after x .

Listing 24.37 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.

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.

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

Prototype `#include <math.h>`
`double remainder (double x, double y);`

Parameters Parameters for this function are:

<code>x</code>	<code>double</code>	The first value
<code>y</code>	<code>double</code>	The second value

Return The remainder $x \text{ REM } y$

See Also [“remquo” on page 230](#)

Listing 24.38 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 remquo function is not implemented on any platform. The function remquo computes the same remainder as the remainder function.

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 >= 3.

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

Prototype

```
#include <math.h>
double remquo (double x, double y, int *quo);
```

Parameters Parameters for this function are:

x	double	First value
y	double	Second value
quo	int*	Pointer to an object quotient

Return The remainder of x and y.

Remarks The value of x may be so large in magnitude relative to y that an exact representation of the quotient is not practical.

See Also [“remainder” on page 229](#)

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.

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

Prototype

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

Parameters Parameters for this function are:

	x	double	The value to be computed
Return			The argument in integral value in floating point format.
See Also			“rinttol” on page 231

Listing 24.39 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

The rint function is not implemented on any platform. Rinttol rounds its argument to the nearest integral value using the current rounding direction.

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

Prototype `#include <math.h>
long int rinttol (double x);`

Parameters Parameters for this function are:

x double Value being rounded

Return The argument in integral value in floating point format.

Remarks If the rounded range is outside the range of long, result is unspecified

See Also [“rint” on page 230](#)

round

Round rounds its argument to an integral value in floating-point format. Rounding halfway cases away from zero, regardless of the current rounding direction.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <math.h> double round (double x);
Parameters	Parameters for this function are:
	x double The value to be rounded
Return	The argument rounded to an integral value in floating point format nearest to but no larger in magnitude than the argument.
See Also	“roundtol” on page 232

Listing 24.40 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 roundtol function is not implemented on any platform.

The function roundtol rounds its argument to the nearest integral value. Rounding halfway cases away from zero, regardless of the current rounding direction.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <math.h> long int roundtol (double round);
Parameters	Parameters for this function are:
	round double The value being rounded
Return	The argument rounded to an integral value in long int format.
Remarks	If the rounded range is outside the range of long, result is unspecified
See Also	“round” on page 232

scalb

The function scalb computes $x * \text{FLT_RADIX}^n$ efficiently, not normally by computing FLT_RADIX^n explicitly.

The scalb function is not implemented on any platform.

Prototype	#include <math.h> double scalb (double x, long int n);
Parameters	Parameters for this function are:
	x double The original value
	n long int Power value
Return	$x * \text{FLT_RADIX}^n$
Remarks	A range error may occur

trunc

Trunc rounds its argument to an integral value in floating-point format nearest to but no larger in magnitude than the argument.

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

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

NOTE For 68k processors returns an integral value.

Parameters Parameters for this function are:

 x double The value to be truncated.

Return The argument to an integral value in floating-point format.

Listing 24.41 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 235](#) begins a thread
 - [“_beginthreadex” on page 236](#), begins a with local data.
 - [“_endthread” on page 237](#) ends thread for `_beginthread`
 - [“_endthreadex” on page 237](#), ends thread for
`_beginthreadex`

_beginthread

This function starts a thread for a multi-threaded application.

Compatibility	A Windows only function.									
Prototype	<pre>#include <process.h> long _beginthread (void (__cdecl *inCodeAddress) (void *)) thread, unsigned int inStackSize, void *inParameter;</pre>									
Parameters	<p>Parameters for this facility are:</p> <table><tr><td>thread</td><td>void *</td><td>The address of the thread function</td></tr><tr><td>inStackSize</td><td>int</td><td>Set by the linkers /STACK switch, 1MB is the default</td></tr><tr><td>inParameter</td><td>void *</td><td>The same as the lpvThreadParameter originally passed, used to pass an initialization routine.</td></tr></table>	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.
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.
Return	.The thread handle is returned or zero upon failure.
See Also	“endthread” on page 237 “beginthreadex” on page 236

_beginthreadex

Begins a thread.

Compatibility A Windows only function.

Prototype

```
#include <process.h>
HANDLE __cdecl _beginthreadex(
    LPSECURITY_ATTRIBUTES inSecurity,
    DWORD inStackSize,
    LPTHREAD_START_ROUTINE inCodeAddress,
    LPVOID inParameter,
    DWORD inCreationFlags,
    LPDWORD inThreadID);
```

Parameters Parameters for this function are:

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.

Return	A HANDLE variable if successful.
Remarks	The function _beginthreadex is similar to the Windows call CreateThread except this functions properly creates the local data used by MSL.
See Also	“_endthreadex” on page 237

_endthread

This function ends a thread called by `_beginthread`.

Compatibility Windows only compatible function.

Prototype `#include <process.h>`
`void _endthread(void);`

Parameters This facility has no parameters.

Return There is no return value for this function.

See Also [“_beginthread” on page 235](#)
[“_endthreadex” on page 237](#)

_endthreadex

Exits the thread.

Compatibility A Windows only function.

Prototype `#include <process.h>`
`VOID __cdecl _endthreadex(DWORD inReturnCode);`

Parameters Parameters for this function are:

 inReturnCode DWORD The exit code is passed through this argument.

Return None, the thread is over.

Remarks The function _endthreadex is similar to the Windows call ExitThread except this functions properly destroys the thread local data used by MSL.

See Also [“_beginthreadex” on page 236](#)

Process.h

Overview of Process.h

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 241](#) 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 240](#) 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()`.

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

Prototype

```
#include <setjmp.h>
void longjmp(jmp_buf env, int val);
```

Parameters Parameters for this facility are:

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.

NOTE 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.

Returns 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.

NOTE The `env` variable must be initialized by a previously executed `setjmp()` before being used by `longjmp()` to avoid undesired results in program execution.

See Also

[“`setjmp`” on page 241](#)

[“`signal`” on page 247](#)

[“`abort`” on page 401](#)

Listing 26.1 For example of longjmp() usage

[“setjmp\(\) example” on page 241](#).

setjmp

Save the processor state for longjmp().

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

Prototype `#include <setjmp.h>`
`int setjmp(jmp_buf env);`

Parameters Parameters for this facility are:

`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()`.

NOTE 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.

Return 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.

See Also [“longjmp” on page 240](#)
[“signal” on page 247](#)
[“abort” on page 401](#)

Listing 26.2 setjmp() example

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

setjmp.h

Overview of setjmp.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-signaled interrupts, segment violation, and program termination.

- These signals, described in [“signal.h Signal descriptions” on page 246](#), are defined as macros in the `signal.h` file.
- [“signal” on page 247](#) 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 249](#) calls the signal handling function
- [“Signal function handling arguments” on page 247](#) 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 249](#), and, in the case of the `SIGABRT` signal, through the function [“abort” on page 401](#). 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: 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.

Table 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.

Macro	Description
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 exit() 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 247](#) describes the pre-defined signal handling macros.

Table 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

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

Prototype

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

Parameters Parameters for this facility are:

	sig	int	A number associated with the signal handling function
	func	void *	A pointer to a signal handling function
Remarks	The <code>signal()</code> function returns a pointer to a signal handling routine that takes an <code>int</code> value argument.		
			The <code>sig</code> argument is the signal number associated with the signal handling function. The signals defined in <code>signal.h</code> are listed in “signal.h Signal descriptions” on page 246 .
			The <code>func</code> 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 247 .
			When it is raised, a signal handler's execution is preceded by the invocation of <code>signal(sig, SIG_DFL)</code> . This call to <code>signal()</code> effectively disables the user's handler. It can be reinstalled by placing a call within the user handler to <code>signal()</code> with the user's handler as its function argument.
Return	<code>signal()</code> returns a pointer to the signal handling function set by the last call to <code>signal()</code> for signal <code>sig</code> . If the request cannot be honored, <code>signal()</code> returns <code>SIG_ERR</code> .		
See Also	“raise” on page 249 “abort” on page 401 “atexit” on page 403 “exit” on page 415		

Listing 27.1 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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <signal.h> int raise(int sig);
Parameters	Parameters for this facility are:
	sig int A signal handling function
Remarks	The <code>raise()</code> function calls the signal handling function associated with signal <code>sig</code> .
Return	<code>raise()</code> returns a zero if the signal is successful; it returns a nonzero value if it is unsuccessful.
See Also	“longjmp” on page 240 “signal” on page 247 “abort” on page 401 “atexit” on page 403 “exit” on page 415

Listing 27.2 For example of `raise()` usage

Refer to the example for [“Example of `signal\(\)` usage” on page 248](#)

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 251](#) A general description of SIOUX properties.
- [“SIOUX for Macintosh” on page 252](#) 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” on page 27](#), 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.

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

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.

See Also [“Overview of console.h” on page 49.](#)

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 253](#) shows a running SIOUX program.
- [“Customizing SIOUX” on page 255](#) shows how to customize your SIOUX window.
 - [“The SIOUXSettings structure” on page 256](#) list structure members that may be set for altering SIOUX's appearance
- [“Using SIOUX windows in your own application” on page 261](#) contains information for using Mac OS facilities with in your SIOUX project.
 - [“path2fss” on page 262](#) a function similar to PBMakeFSSpec.
 - [“SIOUXHandleOneEvent” on page 263](#) allows you to use an even in SIOUX

- [“SIOUXSetTitle” on page 264](#) 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-sizable, 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 260](#)

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 261](#).

Creating a Project with SIOUX

To use the SIOUX library, create a project from a project stationery pads that creates an Console style project.

NOTE 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 `printf()` and `scanf()` 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 255](#).

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 261](#).

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 C++.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_C++_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_C++_Mach-O.lib (for C++)
- MSL_SIOUX_Mach-O.lib (to be implemented)
- MSL_Runtime_Mach-O.lib

Customizing SIOUX

This 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 `SIOUXSettings` before you call any function that uses standard input or output. If you modify `SIOUXSettings` afterwards, SIOUX does not change its window.

The first three sections, “[Changing the font and tabs](#)” on page 258, “[Changing the size and location](#)” on page 260, and “[Showing the status line](#)” on page 261, describe how to customize the SIOUX window. The next section, “[Changing what happens on quit](#)” on page 260, describe how to modify how SIOUX acts when you quit it. The last section, “[Using SIOUX windows in your own application](#)”

[on page 261](#), describes how you can use a SIOUX window in your own Macintosh program.

[“The SIOUXSettings structure” on page 256](#) summarizes what’s in the SIOUXSettings structure.

Table 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.
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	The location of the top of the SIOUX window.
short	leftpixel	The location of the left of the SIOUX window.

This field...		Specifies...
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 257](#) contains a small program that customizes a SIOUX window.

Listing 28.1 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 `tabspace` 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 `fontid`, `fontsize`, and `fontface`. If that font is too large, the status line may be unreadable. You can remove the status line by setting the field `showstatusline` to `false`, as described in [“Showing the status line” on page 261](#).

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 tabspace controls how SIOUX handles tabs. If tabspace 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 tabspace 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 tabspace to an appropriate number, such as 4 or 8. By default, tabspace 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`.

TIP 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 261](#).

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`.

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

Prototype

```
#include <path2fss.h>
OSErr __path2fss
    (const char * pathName, FSSpecPtr spec)
```

Parameters Parameters for this facility are:

	pathname	const char *	The path name
	spec	FSSpecPtr	A file specification pointer
Remarks	This function is similar to PBMakeFSSpec with three major differences:		
	<ul style="list-style-type: none">• 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 PBMakeFSSpec, 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.		
Return	Errors are returned for invalid path names or path names that specify directories rather than files.		
See Also	“Inside Macintosh: Files”		

SIOUXHandleOneEvent

Handles an event for a SIOUX window.

Compatibility	Macintosh only, This function may not be implemented on all Mac OS versions.
Prototype	<pre>#include <SIOUX.h> Boolean SIOUXHandleOneEvent(EventRecord *event);</pre>
Parameters	Parameters for this facility are:
	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.

Return If it handles the event, SIOUXHandleOneEvent () returns true.
Otherwise, SIOUXHandleOneEvent () returns false.

Listing 28.2 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.

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

Prototype `include <SIOUX.h>`
`extern void SIOUXSetTitle`
`(unsigned char title[256])`

Parameters Parameters for this facility are:

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.

Return There is no return value from `SIOUXSetTitle()`

Listing 28.3 Example of `SIOUXSetTitle()` usage.

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

int main(void)
{
    printf("Hello World\n");
    SIOUXSetTitle("\pMy Title");

    return 0;
}
```

SIOUX.h*SIOUX for Macintosh*

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 270](#) to change a files attributes
- [“fstat” on page 271](#) to get information on an open file
- [“mkdir” on page 272](#) to make a directory for folder
- [“stat” on page 273](#) 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” on page 27](#), 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 268](#). It also has several type definitions and file mode definitions. The necessary types are listed in [“Defined types” on page 268](#). The file modes are

listed in “[File Modes](#)” on page 269. File mode macros are listed in “[File Mode Macros Non Windows](#)” on page 269 and “[File Mode Macros Windows Only](#)” on page 270.

Table 29.1 **Defined types**

Type	Used to Store
dev_t	Device type
gid_t	The file size
ino_t	File information
mode_t	File Attributes
nlink_t	The number of links
off_t	The file size in bytes
uid_t	The user's ID

Table 29.2 **The stat or _stat Structure**

Type	Variable	Purpose
mode_t	st_mode	File mode, see “ File Modes ” on page 269
ino_t	st_ino	File serial number
dev_t	st_dev	ID of device containing this file
dev_t	std_rdev (Windows)	ID of device containing this file
nlink_t	st_nlink	Number of links
uid_t	st_uid	User ID of the file's owner
gid_t	st_gid	Group ID of the file's group
off_t	st_size	File size in bytes
_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

Type	Variable	Purpose
long	st_blksize	Optimal blocksize
long	st_blocks	Blocks allocated for file

File Modes

File mode information.

Table 29.3 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)

Table 29.4 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_IRWXO	Permissions for file other class
S_IRWXU	Permissions for file owner class
S_ISGID	Set group ID on execution

File Mode	Purpose
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

Table 29.5 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.

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

Prototype

```
#include <stat.h>
int chmod(const char *, mode_t);
int _chmod(const char *, mode_t);
```

Parameters Parameters for this facility are:

path	const char *	The file to change modes on
mod	mode_t	A mask of the new file attributes

Return .The file attributes as a mask is returned or a negative one on failure.

See Also [“fstat” on page 271](#)

fstat

Gets information about an open file.

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

Prototype

```
#include <stat.h>
int fstat(int fildes, struct stat *buf);
int _fstat(int fildes, struct stat *buf);
```

Parameters Parameters for this facility are:

 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 267](#).

Return If it is successful, `fstat()` returns zero. If it encounters an error, `fstat()` returns -1 and sets `errno`.

See Also [“stat” on page 273](#)

Listing 29.1 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.

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

Prototype `#include <stat.h>`
`int mkdir(const char *path, int mode);`
`int _mkdir(const char *path);`

Parameters Parameters for this facility are:

<code>path</code>	<code>const char *</code>	The path name
<code>mode</code>	<code>int</code>	The open mode (Not applicable for Windows)

Remarks This function creates the new folder specified in `path`. It ignores the argument `mode`.

Return If it is successful, `mkdir()` returns zero. If it encounters an error, `mkdir()` returns -1 and sets `errno`.

See Also [“unlink” on page 519](#)
[“rmdir” on page 513](#)

Listing 29.2 Example for mkdir()

Macintosh

```
#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;
}
```

Creates a folder named Asok as a sub-folder of the current folder

stat

Gets information about a file.

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

Prototype `#include <stat.h>`
`int stat(const char *path, struct stat *buf);`

```
int _stat(const char *path, struct stat *buf);
```

Parameters Parameters for this facility are:

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 267.](#)

Return If it is successful, stat() returns zero.

See Also [“fstat” on page 271](#)

[“uname” on page 533](#)

Listing 29.3 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.

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

Prototype

```
#include <stat.h> /* Macintosh */
mode_t umask(mode_t cmask)
mode_t _umask(mode_t cmask)
```

Parameters Parameters for this facility are:

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` 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.

Return 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`.

stat.h

Overview of stat.h

See Also

[“creat” on page 124](#)

[“open” on page 126](#)

[“mkdir” on page 272](#)

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 278](#) a variable argument list
- [“`va_copy`” on page 278](#)
- [“`va_end`” on page 279](#) a variable argument end
- [“`va_start`” on page 279](#) 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 279](#) initializes the `va_list` variable and is called before gaining access to the arguments. The macro [“`va_arg`” on page 278](#) returns each of the arguments in `va_list`. When all the arguments have been processed through `va_arg()`, the macro [“`va_end`” on page 279](#) is called to allow a normal return from the function.

va_arg

Macro to return an argument value.

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

Prototype

```
#include <stdarg.h>
type va_arg(va_list ap, type type);
```

Parameters Parameters for this facility are:

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.

Return The `va_arg()` macro returns the next argument on the function's argument list of `type`.

See Also [“va_end” on page 279](#)
[“va_start” on page 279](#)

Listing 30.1 For example of va() usage

Refer to the example [“Example of va_start\(\) usage.” on page 280](#).

va_copy

Copies and initializes a variable argument list.

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

Prototype

```
#include <stdarg.h>
void va_copy(va_list dest, va_list src)
```

Parameters Parameters for this facility are:

	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		
Return	There is no return for this facility.		
See Also	“Variable arguments for functions” on page 277		

va_end

Prepare a normal function return.

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

Prototype

```
#include <stdarg.h>
void va_end(va_list ap);
```

Parameters Parameters for this facility are:

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.

See Also [“va_arg” on page 278](#)
[“va_start” on page 279](#)

Listing 30.2 For example of va_end usage

Refer to the example [“Example of va_start\(\) usage.” on page 280](#).

va_start

Initialize the variable-length argument list.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	#include <stdarg.h> void va_start(va_list ap, ParmN Parm);		
Parameters	Parameters for this facility are: 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.		
See Also	“va_arg” on page 278 “va_end” on page 279		

Listing 30.3 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 on page 283](#) only if the compiler support for c99 has been turned on using the C99 pragma.

```
#pragma c99 on | off | reset
```

Table 31.1 The header `stdbool.h` Boolean Defines

<code>bool</code>	<code>_Bool</code>
<code>true</code>	<code>1</code>
<code>false</code>	<code>0</code>
<code>__bool_true_false_are_defined</code>	<code>1</code>

There are no other defines in this header.

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 macros and types are defined in `stddef.h`

- [“NULL” on page 285](#), defines NULL
- [“offsetof” on page 285](#), is the offset of a structure’s member
- [“ptrdiff_t” on page 286](#), used for pointer differences
- [“size_t” on page 286](#), is the return from a sizeof operation
- [“wchar_t” on page 286](#), is a wide character type

Commonly used definitions

The `stddef.h` header file defines commonly used macros and types that are used throughout the ANSI C Standard Library.

Compatibility

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

NULL

The `NULL` macro is the null pointer constant used in the Standard Library.

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.

ptrdiff_t

The `ptrdiff_t` type is the signed integral type used for holding the result of subtracting one pointer's value from another.

size_t

The `size_t` type is an unsigned integral type returned by the `sizeof()` operator.

wchar_t

The `wchar_t` type is an integral type capable of holding all character representations of the wide character set.

stdint.h

The header `stdbool.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 287](#)
- [“Limits of specified-width integer types” on page 289](#)
- [“Macros for integer constants” on page 293](#)
- [“Macros for greatest-width integer constants” on page 293](#)

Compatibility

This 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 288](#)
- [“Minimum-width integer types” on page 288](#)
- [“Fastest minimum-width integer types” on page 288](#)
- [“Integer types capable of holding object pointers” on page 288](#)
- [“Greatest-width integer types” on page 289](#)
- [“Mac OS X Specific Integer Types” on page 289](#)

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 types

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	__std(int32_t)		

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 290](#)
- [“Maximum Values Of Exact-width Signed Integer Types” on page 290](#)
- [“Maximum Values Of Exact-width Unsigned Integer Types” on page 290](#)
- [“Minimum Values Of Minimum-width Signed Integer Types” on page 290](#)
- [“Maximum Values Of Minimum-width Signed Integer Types” on page 291](#)
- [“Maximum Values Of Minimum-width Unsigned Integer Types” on page 291](#)
- [“Minimum Values Of Fastest Minimum-width Signed Integer” on page 291](#)
- [“Maximum Values Of Fastest Minimum-width Signed Integer Types” on page 291](#)
- [“Maximum Values Of Fastest Minimum-width Unsigned Integer Types” on page 292](#)
- [“Minimum Value Of Pointer-holding Signed Integer Type” on page 292](#)

- “[Minimum Value Of Greatest-width Signed Integer Type](#)” on [page 292](#)
- “[Maximum Value Of Greatest-width Signed Integer Type](#)” on [page 292](#)
- “[Limits Of Other Integer Types](#)” on [page 292](#)

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_LEAST8_MAX	UCHAR_MAX
UINT_LEAST16_MAX	USHRT_MAX
UINT_LEAST32_MAX	ULONG_MAX
UINT_LEAST64_MAX	ULLONG_MAX

Table 33.13 Minimum Values Of Fastest Minimum-width Signed Integer

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 Type

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.

Prototype	INT8_C(value)	value
Prototype	INT16_C(value)	value
Prototype	INT32_C(value)	value ## L
Prototype	INT64_C(value)	value ## LL
Prototype	UINT8_C(value)	value ## U
Prototype	UINT16_C(value)	value ## U
Prototype	UINT32_C(value)	value ## UL
Prototype	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`.

Prototype	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`.

Prototype	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 301](#) clears an error from a stream
- [“fclose” on page 303](#) closes a file
- [“fdopen” on page 305](#), converts a file descriptor to a stream
- [“ferror” on page 307](#) checks a file error status
- [“fflush” on page 309](#) flushes a stream
- [“fgetc” on page 311](#) gets a character from a file
- [“fgetpos” on page 313](#) gets a file position from large files
- [“fgets” on page 315](#) gets a string from a file
- [“_fileno” on page 316](#) Windows version only
- [“fopen” on page 316](#) opens a file
- [“fprintf” on page 319](#) prints formatted output to a file
- [“fputc” on page 327](#) writes a character to a file
- [“fputs” on page 328](#) writes a string to a file
- [“fread” on page 330](#) reads a file
- [“freopen” on page 332](#) reopens a file
- [“fscanf” on page 333](#) scans a file

- [“fseek” on page 340](#) moves to a file position
- [“fsetpos” on page 342](#) sets a file position for large files
- [“ftell” on page 343](#) tells a file offset
- [“fwide” on page 344](#) determines a character orientation
- [“fwrite” on page 346](#) writes to a file
- [“getc” on page 347](#) gets a character from a stream
- [“getchar” on page 349](#) gets a character from stdin
- [“gets” on page 350](#) gets a string from stdin
- [“perror” on page 352](#) writes an error to stderr
- [“printf” on page 353](#) writes a formatted output to stdout
- [“putc” on page 361](#) writes a character to a stream
- [“putchar” on page 363](#) writes a character to stdout
- [“puts” on page 364](#) writes a string to stdout
- [“remove” on page 365](#) removes a file
- [“rename” on page 366](#) renames a file
- [“rewind” on page 368](#) resets the file indicator to the beginning
- [“scanf” on page 369](#) scans stdin for input
- [“setbuf” on page 375](#) sets the buffer size for a stream
- [“setvbuf” on page 376](#) sets the stream buffer size and scheme
- [“snprintf” on page 378](#) writes a number of characters to a buffer
- [“sprintf” on page 379](#) to write to a character buffer
- [“sscanf” on page 381](#) to scan a string
- [“tmpfile” on page 382](#) to create a temporary file
- [“ungetc” on page 385](#) to place a character back in a stream
- [“vfprintf” on page 386](#) write variable arguments to file
- [“vfscanf” on page 389](#) a variable argument scanf
- [“vprintf” on page 391](#) write variable arguments to stdout
- [“vsnprintf” on page 392](#) write variable arguments to a char array buffer with a number limit
- [“vsprintf” on page 394](#) write variable arguments to a char array buffer
- [“vsscanf” on page 396](#) A variable scanf

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 298](#), 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>vscanf</code>	<code>vprintf</code>

Table 34.2 The Wide-Character Oriented Functions in Wchar.h are:

fgetwc	fgetws	fwprintf	fputwc	fputws	fwscanf
getwc	getwchar	putwc	putwchar	swprintf	swscanf
towctrans	vfwscanf	vswscanf	vwscanf	vfwprintf	vswprintf
vwprintf	wasctime	watof	wcsat	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

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.

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

Prototype

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

Parameters Parameters for this facility are:

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.

See Also

[“feof” on page 306](#)
[“ferror” on page 307](#)
[“fopen” on page 316](#)
[“fseek” on page 340](#)
[“rewind” on page 368](#)

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.

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

Prototype `#include <stdio.h>`
`int fclose(FILE *stream);`

Parameters Parameters for this facility are:

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()`.

NOTE On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return `fclose()` returns a zero if it is successful and returns an EOF if it fails to close a file.

See Also [“fopen” on page 316](#)
[“freopen” on page 332](#)
[“tmpfile” on page 382](#)
[“abort” on page 401](#)
[“exit” on page 415](#)

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.

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

Prototype

```
#include <stdio.h>
FILE *fdopen(int fildes, char *mode);
FILE *_fdopen(int fildes, char *mode);
```

Parameters Parameters for this facility are:

fildes	int	A file descriptor, which is integer file number that can be obtained from the function <code>fileno()</code> .
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.

Return If it is successful, `fdopen()` returns a stream. If it encounters an error, `fdopen()` returns `NULL`.

See Also [“fileno” on page 91](#)
[“open” on page 126](#)

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.

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

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

Parameters Parameters for this facility are:

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.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return `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.

See Also [“clearerr” on page 301](#)
[“ferror” on page 307](#)

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.

Compatibility

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

Prototype `#include <stdio.h>`
 `int ferror (FILE *stream);`

Parameters Parameters for this facility are:

 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.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return `ferror()` returns a nonzero value if `stream`'s error status is on, and returns zero if `stream`'s error status is off.

See Also [“clearerr” on page 301](#)
[“feof” on page 306](#)

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.

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

Prototype `#include <stdio.h>`
`int fflush(FILE *stream);`

Parameters Parameters for this facility are:

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.

NOTE The `fflush()` function should not be used after an input operation.

NOTE Using `fflush()` for input streams especially the standard input stream (`stdin`) is undefined and is not supported and will not flush the input buffer.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return The function `fflush()` returns `EOF` if a write error occurs, otherwise it returns zero.

See Also [“setvbuf” on page 376](#)

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.

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

Prototype `#include <stdio.h>
int fgetc(FILE *stream);`

Parameters Parameters for this facility are:

stream FILE * A pointer to a FILE stream

Remarks The `fgetc()` function reads the next character from `stream` and advances its file position indicator.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return `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.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“getc” on page 347](#)[“getchar” on page 349](#)**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.

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

Prototype

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);
```

Parameters Parameters for this facility are:

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.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return `fgetpos()` returns zero when successful and returns a nonzero value when it fails.

See Also [“fseek” on page 340](#)

[“fsetpos” on page 342](#)

[“ftell” on page 343](#)

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.

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

Prototype `#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);`

Parameters Parameters for this facility are:

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.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return `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.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“gets” on page 350](#)

Listing 34.9 For example of fgets() usage

Refer to [“Example of feof\(\) usage.” on page 307](#) for `feof()`.

_fileno

This function is described in `extras.h` as [“_fileno” on page 91](#) in this header it is Windows only.

fopen

Open a file as a stream.

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

Prototype

```
#include <stdio.h>
FILE *fopen(const char *filename,
            const char *mode);
```

Parameters Parameters for this facility are:

filename	const char *	The filename of the file to open
mode	const char *	The file opening mode

Remarks	<p>The <code>fopen()</code> function opens a file specified by <code>filename</code>, and associates a stream with it. The <code>fopen()</code> function returns a pointer to a <code>FILE</code>. This pointer is used to refer to the file when performing I/O operations.</p> <p>The mode argument specifies how the file is to be used. “Open modes for fopen()” describes the values for mode.</p> <p>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 <code>fflush()</code> function or one of the file positioning operations (<code>fseek()</code>, <code>fsetpos()</code>, or <code>rewind()</code>). 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.</p> <p>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.</p>
NOTE	Write modes, even if in Write and Read (w+, wb+) delete any current data in a file when the file is opened.

Table 34.3 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)

Mode	Description
“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.
“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)

Return `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.

See Also [“fclose” on page 303](#)

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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <stdio.h> int fprintf(FILE *stream, const char *format, ...);</pre>
Parameters	Parameters for this facility are: stream FILE * A pointer to a FILE stream format const char * The format string
Remarks	The <code>fprintf()</code> function writes formatted text to <code>stream</code> and advances the file position indicator. Its operation is the same as <code>printf()</code> with the addition of the <code>stream</code> argument. Refer to the description of <code>printf()</code> . 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 <code>fflush()</code> function or one of the file positioning operations (<code>fseek()</code> , <code>fsetpos()</code> , or <code>rewind()</code>).
NOTE	On embedded/ RTOS systems this function only is implemented for <code>stdin</code> , <code>stdout</code> and <code>stderr</code> 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 322](#) 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).

NOTE 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 322](#) 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 360](#) demonstrates their use.

Table 34.4 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, 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	AltiVec: A vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X
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

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 (<code>float</code> or <code>double</code>) is output in scientific notation: <code>[-]b.aaaee±Eee</code> . There is one digit (<i>b</i>) 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.

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.
----	---

Return `fprintf()` returns the number of arguments written or a negative number if an error occurs.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“printf” on page 353](#)
[“sprintf” on page 379](#)
[“vfprintf” on page 386](#)
[“vprintf” on page 391](#)
[“vsprintf” on page 394](#)

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.

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

Prototype

```
#include <stdio.h>
int fputc(int c, FILE *stream);
```

Parameters Parameters for this facility are:

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()`).

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return `fputc()` returns the character written if it is successful, and returns `EOF` if it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“putc” on page 361](#)

[“putchar” on page 363](#)

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:

ABCDEFGHIJKLMNPQRSTUVWXYZ

fputs

Write a character array to a stream.

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

Prototype `#include <stdio.h>`
`int fputs(const char *s, FILE *stream);`

Parameters Parameters for this facility are:

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()).

NOTE On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return fputs() returns a zero if successful, and returns a nonzero value when it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“puts” on page 364](#)

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.

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

Prototype `#include <stdio.h>
size_t fread(void *ptr, size_t size,
 size_t nmemb, FILE *stream);`

Parameters Parameters for this facility are:

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.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return

`fread()` returns the number of items read successfully.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“fgets” on page 315](#)

[“fwrite” on page 346](#)

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.

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

Prototype `#include <stdio.h>
FILE *freopen(const char *filename,
 const char *mode, FILE *stream);`

Parameters Parameters for this facility are:

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.

Return `fopen()` returns the value of `stream`, if it is successful. If `fopen()` fails it returns a null pointer (`NULL`).

See Also [“fopen” on page 316](#)

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.

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

Prototype `#include <stdio.h>
int fscanf(FILE *stream,
 const char *format, ...);`

Parameters Parameters for this facility are:

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.

NOTE 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 Functions” on page 335](#), 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 374](#) demonstrates their use.

Table 34.5 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.
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	Altivec: vector short, vector unsigned short, vector bool short or vector pixel
hv	when followed by c, d, i, o, u, x or X
vl	Altivec: vector long, vector unsigned long
lv	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().

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 scanset. See " "Scanset" on page 337 , 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.

Return `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.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“errno” on page 83](#)

[“scanf” on page 369](#)

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%f\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.

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

Prototype

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

Parameters Parameters for this facility are:

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 298.

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`.

NOTE On embedded/RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return `fseek()` returns zero if it is successful and returns a nonzero value if it fails.

See Also [“fgetpos” on page 313](#)
[“fsetpos” on page 342](#)
[“ftell” on page 343](#)

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.

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

Prototype `#include <stdio.h>`
`int fsetpos(FILE *stream, const fpos_t *pos);`

Parameters Parameters for this facility are:

	stream FILE * A pointer to a FILE stream
	pos fpos_t A pointer to a file positioning type
Remarks	The <code>fsetpos()</code> function sets the file position indicator for <code>stream</code> using the value pointed to by <code>pos</code> . The function is used in conjunction with <code>fgetpos()</code> when dealing with files having sizes greater than what can be represented by the <code>long int</code> argument used by <code>fseek()</code> .
	<code>fsetpos()</code> undoes the previous call to <code>ungetc()</code> and clears the end-of-file status.
NOTE	On embedded/RTOS systems this function only is implemented for <code>stdin</code> , <code>stdout</code> and <code>stderr</code> files.
Return	<code>fsetpos()</code> returns zero if it is successful and returns a nonzero value if it fails.
See Also	“fgetpos” on page 313 “fseek” on page 340 “ftell” on page 343

Listing 34.18 For example of `fsetpos()` usage

Refer to [“Example of `fgetpos\(\)` usage.” on page 314](#) for `fgetpos()`.

ftell

Return the current file position indicator value.

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

Prototype
`#include <stdio.h>`
`long int ftell(FILE *stream);`

Parameters
Parameters for this facility are:

stream FILE * A pointer to a FILE stream

Remarks	The <code>ftell()</code> function returns the current value of stream's file position indicator. It is used in conjunction with <code>fseek()</code> 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 <code>stdin</code> , <code>stdout</code> , or <code>stderr</code> , where a file indicator position is not applicable. Also, <code>ftell()</code> cannot handle files with sizes larger than what can be represented with a <code>long int</code> . In such a case, use the <code>fgetpos()</code> and <code>fsetpos()</code> functions.
NOTE	On embedded/ RTOS systems this function only is implemented for <code>stdin</code> , <code>stdout</code> and <code>stderr</code> files.
Return	<code>ftell()</code> , when successful, returns the current file position indicator value. If it fails, <code>ftell()</code> returns <code>-1L</code> and sets the global variable <code>errno</code> to a nonzero value.
See Also	“errno” on page 83. “fgetpos” on page 313

Listing 34.19 For example of `ftell()` usage

Refer to [“Example of `fseek\(\)` usage.” on page 341](#) for `fseek()`.

fwide

Determine the orientation of a stream.

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

Prototype

```
#include <stdio.h>
           int fwide(FILE *stream, int orientation);
```

Parameters Parameters for this facility are:

stream FILE *	A pointer to the stream being tested
orientat int ion	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.

Return 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.

Listing 34.20 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.

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

Prototype `#include <stdio.h>
size_t fwrite(const void *ptr, size_t size,
 size_t nmemb, FILE *stream);`

Parameters Parameters for this facility are:

<code>ptr</code>	<code>void *</code>	A pointer to the item being written
<code>size</code>	<code>size_t</code>	The size of the item being written
<code>nmemb</code>	<code>size_t</code>	The number of items being written
<code>stream</code>	<code>FILE *</code>	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()).

NOTE On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return fwrite() returns the number of items successfully written to stream.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“fread” on page 330](#)

Listing 34.21 For example of fwrite() sage

Refer to [“Example of fread\(\) usage.” on page 331](#) for fread().

getc

Read the next character from a stream.

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

Prototype `#include <stdio.h>`
`int getc(FILE *stream);`

Parameters Parameters for this facility are:

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.

Return `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.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“fgetc” on page 311](#)

[“fputc” on page 327](#)

[“getchar” on page 349](#)

[“putchar” on page 363](#)

Listing 34.22 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`.

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

Prototype `#include <stdio.h>`
`int getchar(void);`

Parameters None

Remarks The `getchar()` function reads a character from the `stdin` stream.

NOTE 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.

Return `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.

See also: [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“fgetc” on page 311](#)

[“getc” on page 347](#)[“putchar” on page 363](#)**Listing 34.23 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`.

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

Prototype `#include <stdio.h>`
`char *gets(char *s);`

Parameters Parameters for this facility are:

`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.

Return `gets()` returns `s` if it is successful and returns a null pointer if it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“fgets” on page 315](#)

Listing 34.24 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`.

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

Prototype `#include <stdio.h>
void perror(const char *s);`

Parameters Parameters for this facility are:

`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`.

See Also [“abort” on page 401](#)
[“errno” on page 83](#)

Listing 34.25 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.

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

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

Parameters Parameters for this facility are:

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 355](#) 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).

NOTE

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 355](#), 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 360](#) demonstrates their use.

Table 34.6 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	AltiVec: A vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X
hv	AltiVec: A vector int, vector unsigned int or vector bool int when followed by c, d, i, o, u, x or X
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.
- 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 (<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.aaaee±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.

Return printf(), like fprintf(), sprintf(), vfprintf(), and vprintf(), returns the number of arguments that were successfully output. printf() returns a negative value if it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“fprintf” on page 319](#)
[“sprintf” on page 379](#)
[“vfprintf” on page 386](#)
[“vprintf” on page 391](#)
[“vsprintf” on page 394](#)

Listing 34.26 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, 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 34.27 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%.@.8lvX\n", " , 0x", vs32);

printf("flt32 = %,5.2vf\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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	#include <stdio.h> int putc(int c, FILE *stream);		
Parameters	Parameters for this facility are: c int The character to write to a file stream FILE * A pointer to a FILE stream		
Remarks	The <code>putc()</code> function outputs <code>c</code> to <code>stream</code> and advances <code>stream</code> 's file position indicator. The <code>putc()</code> works identically to the <code>fputc()</code> 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 <code>fflush()</code> function or one of the file positioning operations (<code>fseek()</code> , <code>fsetpos()</code> , or <code>rewind()</code>).		
Return	<code>putc()</code> returns the character written when successful and return EOF when it fails.		
See Also	“Wide Character and Byte Character Stream Orientation” on page 300 “fputc” on page 327 “putchar” on page 363		

Listing 34.28 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.

Compatibility

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

Prototype `#include <stdio.h>`
 `int putchar(int c);`

Parameters Parameters for this facility are:

 c int The character to write to stdout

Remarks The `putchar()` function writes character `c` to `stdout`.

Return `putchar()` returns `c` if it is successful and returns `EOF` if it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“fputc” on page 327](#)

[“putc” on page 361](#)

Listing 34.29 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.

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

Prototype `#include <stdio.h>`
`int puts(const char *s);`

Parameters Parameters for this facility are:

`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.

Return `puts()` returns zero if successful and returns a nonzero value if it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“fputs” on page 328](#)

Listing 34.30 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.

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

Prototype `#include <stdio.h>`
`int remove(const char *filename);`

Parameters Parameters for this facility are:

filename `const char *` The name of the file to be deleted

Remarks The `remove()` function deletes the named file specified by `filename`.

Return `remove()` returns 0 if the file deletion is successful, and returns a nonzero value if it fails.

See Also [“`fopen`” on page 316](#)
[“`rename`” on page 366](#)

Listing 34.31 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.

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

Prototype `#include <stdio.h>`
`int rename(const char *old, const char *new);`

Parameters Parameters for this facility are:

old	const char *	The old file name
new	const char *	The new file name

Remarks	The <code>rename()</code> function changes the name of a file, specified by <code>old</code> to the name specified by <code>new</code> .
Return	<code>rename()</code> returns a nonzero if it fails and returns zero if successful
See Also	“fopen” on page 332 “remove” on page 365

Listing 34.32 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.

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

Prototype `#include <stdio.h>
void rewind(FILE *stream);`

Parameters Parameters for this facility are:

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.

NOTE On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

See Also [“fseek” on page 340](#)
[“fsetpos” on page 342](#)

Listing 34.33 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.

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

Prototype `#include <stdio.h>`
`int scanf(const char *format, ...);`

Parameters Parameters for this facility are:

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 371](#), 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 374](#) demonstrates their use.

Table 34.7 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.
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	AltiVec: vector short, vector unsigned short, vector bool short or vector pixel
hv	when followed by c, d, i, o, u, x or X
vl	AltiVec: vector long, vector unsigned long
lv	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().
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 337 , for a full description.

Return `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.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“fscanf” on page 333](#)
[“sscanf” on page 381](#)

Listing 34.34 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);
```

stdio.h*Standard input/output*

```
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 34.35 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("abcdefgh", "%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.

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

Prototype `#include <stdio.h>
void setbuf(FILE *stream, char *buf);`

Parameters Parameters for this facility are:

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.

See Also [“setvbuf” on page 376](#)
[“malloc” on page 421](#)

Listing 34.36 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.

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

Prototype `#include <stdio.h>
int setvbuf(FILE *stream, char *buf, int mode,
 size_t size);`

Parameters Parameters for this facility are:

stream	<code>FILE *</code>	A pointer to a FILE stream
buf	<code>char *</code>	A buffer for input and output

	mode	int	A buffering mode
	size	size_t	The size of the buffer
Remarks	The <code>setvbuf()</code> 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 <code>buf</code> argument is a pointer to a character array. The <code>size</code> argument indicates the size of the character array pointed to by <code>buf</code> . The most efficient buffer size is a multiple of <code>BUFSIZ</code> , defined in <code>stdio.h</code> .		
	If <code>buf</code> is a null pointer, then the operating system creates its own buffer of <code>size</code> bytes.		
	The <code>mode</code> argument specifies the buffering scheme to be used with <code>stream</code> . <code>mode</code> can have one of three values defined in <code>stdio.h</code> : <code>_IOFBF</code> , <code>_IOLBF</code> , and <code>_IONBF</code> .		
	<ul style="list-style-type: none"> • <code>_IOFBF</code> specifies that <code>stream</code> be buffered. • <code>_IOLBF</code> specifies that <code>stream</code> be line buffered. • <code>_IONBF</code> specifies that <code>stream</code> be unbuffered 		
Return	<code>setvbuf()</code> returns zero if it is successful and returns a nonzero value if it fails.		
See Also	“setbuf” on page 375 “malloc” on page 421		

Listing 34.37 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.

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

Prototype `#include <stdio.h>`
`int snprintf(char * s, size_t n,`
 `const char * format, ...);`

Parameters Parameters for this facility are:

<code>s</code>	<code>char *</code>	A string to write to
<code>n</code>	<code>size_t</code>	Max number of chars to be written to <code>s</code>
<code>format</code>	<code>const char *</code>	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 320.](#)

Return `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`.

Listing 34.38 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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <stdio.h> int sprintf(char *s, const char *format, ...);
Parameters	Parameters for this facility are: s char * A string to write to format const char * The format string
Remarks	The <code>sprintf()</code> function works identically to <code>printf()</code> with the addition of the <code>s</code> parameter. Output is stored in the character array pointed to by <code>s</code> instead of being sent to <code>stdout</code> . The function terminates the output character string with a null character. For specifications concerning the output control string and conversion specifiers please see: “Output Control String and Conversion Specifiers” on page 320 .
Return	<code>sprintf()</code> returns the number of characters assigned to <code>s</code> , not including the null character.
See Also	“Wide Character and Byte Character Stream Orientation” on page 300 “fprintf” on page 319 “printf” on page 353

Listing 34.39 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.

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

Prototype `#include <stdio.h>
int sscanf(char *s, const char *format, ...);`

Parameters Parameters for this facility are:

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 334](#). Also see [“Scanset” on page 337](#), for a full description of the use of `scansets`.

Return `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.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“fscanf” on page 333](#)
[“scanf” on page 369](#)

Listing 34.40 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 394 %s %d!", s1, s2, s3, &i);
printf("%s %s %s %d\n", s1, s2, s3, i);

return 0;
}
Output:
figs cat road 16
```

tmpfile

Open a temporary file.

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

Prototype `#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.

Return `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`).

See Also [“fopen” on page 316](#)
[“tmpnam” on page 383](#)

Listing 34.41 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.

Compatibility

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

Prototype `#include <stdio.h>`
 `char *tmpnam(char *s);`

Parameters

Parameters for this facility are:

`s` `char *` A temporary file name

Remarks

The `tmpnam()` functions 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.

Return `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.

See Also [“`open`” on page 316](#)
[“`tmpfile`” on page 382](#)

Listing 34.42 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("tempworks") ;

    // 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.

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

Prototype `#include <stdio.h>
int ungetc(int c, FILE *stream);`

Parameters Parameters for this facility are:

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.

Return `ungetc()` returns `c` if it is successful and returns `EOF` if it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)

[“fseek” on page 340](#)

[“fsetpos” on page 342](#)

[“rewind” on page 368](#)

Listing 34.43 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.

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

Prototype

```
#include <stdarg.h>
#include <stdio.h>
int vfprintf(FILE *stream,
              const char *format,va_list arg);
```

Parameters Parameters for this facility are:

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 320](#).

Return vfprintf() returns the number of characters written or EOF if it failed.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“fprintf” on page 319](#)
[“printf” on page 353](#)
[“Overview of stdarg.h” on page 277](#)

Listing 34.44 Example of vfprintf() usage.

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

stdio.h*Standard input/output*

```
#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;
}
```

```
Output to file foo:  
foo 483.5820 56
```

vfscanf

Read formatted text from a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

Prototype

```
#include <stdarg.h>
#include <stdio.h>
int vfscanf(FILE *stream,
             const char *format, va_list arg);
```

Parameters Parameters for this facility are:

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 Functions” on page 335](#).

Return 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.

See Also [“scanf” on page 369](#)
[“fscanf” on page 333](#)

Listing 34.45 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`.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <stdio.h> int vprintf(const char *format, va_list arg);
Parameters	Parameters for this facility are: format const char * The format string arg va_list A variable argument list
Remarks	The <code>vprintf()</code> function works identically to the <code>printf()</code> function. Instead of the variable list of arguments that can be passed to <code>printf()</code> , <code>vprintf()</code> accepts its arguments in the array of type <code>va_list</code> processed by the <code>va_start()</code> macro from the <code>stdarg.h</code> header file. For specifications concerning the output control string and conversion specifiers please see: “Output Control String and Conversion Specifiers” on page 320 .
Return	<code>vprintf()</code> returns the number of characters written or a negative value if it failed.
See Also	“Wide Character and Byte Character Stream Orientation” on page 300 “fprintf” on page 319

[“printf” on page 353](#)

[“Overview of stdarg.h” on page 277](#)

Listing 34.46 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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.												
Prototype	<pre>#include <stdarg.h> #include <stdio.h> int vsnprintf(char * s, size_t n, const char * format, va_list arg);</pre>												
Parameters	Parameters for this facility are:												
	<table><tr><td>s</td><td>char *</td><td>A string to write to</td></tr><tr><td>n</td><td>size_t</td><td>Max number of chars to be written to s</td></tr><tr><td>format</td><td>const char *</td><td>The format string</td></tr><tr><td>arg</td><td>va_list</td><td>A variable argument list</td></tr></table>	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
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 <code>vsnprintf()</code> function works identically to <code>snprintf()</code> , except that the variable list of arguments that can be passed to <code>snprintf()</code> is replaced by an array <code>arg</code> of type <code>va_list</code> , which must have been initialized by the <code>va_start()</code> macro from the <code>stdarg.h</code> header file. The <code>vsnprintf()</code> does not invoke the <code>va_end</code> macro. If <code>n</code> is zero nothing is written; otherwise, any characters beyond the <code>n-1</code> 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 320 .												
Return	<code>Vsnprintf()</code> returns the number of characters that would have been assigned to <code>s</code> , had <code>n</code> 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 <code>n</code> .												
See Also	“Wide Character and Byte Character Stream Orientation” on page 300 “printf” on page 353 “sprintf” on page 379 “Overview of stdarg.h” on page 277												

Listing 34.47 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.

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

Prototype

```
#include <stdio.h>
int vsprintf(char *s,
             const char *format, va_list arg);
```

Parameters Parameters for this facility are:

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 320](#).

Return `vsprintf()` returns the number of characters written to `s` not counting the terminating null character. Otherwise, EOF on failure.

See Also [“Wide Character and Byte Character Stream Orientation” on page 300](#)
[“printf” on page 353](#)
[“sprintf” on page 379](#)
[“Overview of stdarg.h” on page 277](#)

Listing 34.48 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.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

Prototype `#include <stdarg.h>
#include <stdio.h>
int vsscanf(const char * s,
 const char * format, va_list arg);`

Parameters	Parameters for this facility are:	
	s char *	The character string to be scanned
	format char *	The format string
	arg va_list	The variable argument list
Remarks	The <code>vsscanf()</code> function works identically to the <code>sscanf()</code> function. Instead of the variable list of arguments that can be passed to <code>sscanf()</code> , <code>vsscanf()</code> accepts its arguments in the array <code>arg</code> of type <code>va_list</code> , which must have been initialized by the <code>va_start()</code> macro from the <code>stdarg.h</code> header file. The <code>vfscanf()</code> does not invoke the <code>va_end</code> macro.	
	On embedded/RTOS systems this function only is implemented for <code>stdin</code> , <code>stdout</code> and <code>stderr</code> files.	
	For specifications concerning the output control string and conversion specifiers please see: “Length Modifiers And Conversion Specifiers For Formatted Input Functions” on page 335 .	
Return	<code>vfscanf()</code> 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, <code>vfscanf()</code> returns EOF.	
See Also	“scanf” on page 369 “fscanf” on page 333	

Listing 34.49 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,
```

stdio.h*Standard input/output*

```
// respectively
ssc(in, "%s %s pear 394 %s %d!", s1, s2, s3, &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
```

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 for string conversion, pseudo-random number generation, memory management, environment communication, searching and sorting, multibyte character conversion, and integer arithmetic.

The string conversion functions are

- [“atof” on page 405](#)
- [“atoi” on page 406](#)
- [“atol” on page 407](#)
- [“strtod” on page 428](#)
- [“strtof” on page 430](#)
- [“strtol” on page 431](#)
- [“strtold” on page 434](#)
- [“strtoul” on page 436](#)
- [“strtoull” on page 438](#)

The pseudo-random number generation functions are

- [“rand” on page 425](#)
- [“srand” on page 428](#)

The memory management functions are

- [“calloc” on page 412](#)
- [“free” on page 416](#)
- [“malloc” on page 421](#)
- [“realloc” on page 427](#)
- [“vec_calloc” on page 439](#)
- [“vec_free” on page 440](#)
- [“vec_malloc” on page 441](#)
- [“vec_realloc” on page 441](#)

The environment communication functions are

- [“abort” on page 401](#)
- [“atexit” on page 403](#)
- [“exit” on page 415](#)
- [“getenv” on page 417](#)
- [“putenv” on page 424](#)
- [“system” on page 439](#)

The searching and sorting functions are

- [“bsearch” on page 408](#)
- [“qsort” on page 424](#)

The multibyte conversion functions convert UTF-8 multibyte characters to wchar_t type characters (defined in stddef.h). The functions are

- [“mblen” on page 421](#)
- [“mbstowcs” on page 422](#)
- [“mbtowc” on page 423](#)
- [“wcstombs” on page 442](#)
- [“wctomb” on page 443](#)

The integer arithmetic functions are

- [“abs” on page 402](#)
- [“div” on page 414](#)

- [“labs” on page 418](#)
- [“llabs” on page 419](#)
- [“ldiv” on page 419](#)

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.

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

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

Parameters None

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.

See Also [“assert” on page 37](#)
[“longjmp” on page 240](#)
[“raise” on page 249](#)
[“signal” on page 247](#)
[“atexit” on page 403](#)
[“exit” on page 415](#)

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.

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

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

Parameters Parameters for this facility are:

i int The value being computed

Return `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.

See Also [“fabs” on page 192](#)
[“labs” on page 418](#)

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.

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

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

Parameters Parameters for this facility are:

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.

Return `atexit()` returns a zero when it succeeds in installing a new exit function and returns a nonzero value when it fails.

See Also [“exit” on page 415](#)

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.

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

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

Parameters Parameters for this facility are:

 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`.

Return `atof()` returns a floating point value of type `double`.

See Also

[“atoi” on page 406](#)
[“atol” on page 407](#)
[“errno” on page 83](#)
[“strtod” on page 428](#)
[“scanf” on page 369](#)

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.

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

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

Parameters Parameters for this facility are:

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`.

Return `atoi()` returns an integer value of type `int`.

See Also [“atof” on page 405](#)
[“atol” on page 407](#)
[“errno” on page 83](#)
[“strtol” on page 431](#)
[“scanf” on page 369](#)

atol

Convert a character string to a value of type `long`.

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

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

Parameters Parameters for this facility are:

nptr const char * 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`.

Return `atol()` returns an integer value of type `long int`.

See Also [“atof” on page 405](#)
 [“atoi” on page 406](#)
 [“errno” on page 83](#)
 [“strtol” on page 431](#)
 [“scanf” on page 369](#)

bsearch

This function uses the binary search algorithm to make an efficient search of a sorted array for an item.

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

Prototype

```
#include <stdlib.h>
void *bsearch(const void *key, const void *base,
              size_t num, size_t size,
              int (*compare) (const void *, const void
                  *))
```

Parameters Parameters for this facility are:

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.

Return `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`).

See Also [“qsort” on page 424](#)

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 *);
```

stdlib.h

Overview of stdlib.h

```
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++;
            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.

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

Prototype `#include <stdlib.h>
void *calloc(size_t nmemb, size_t elemsize);`

Parameters Parameters for this facility are:

<code>nmemb</code>	<code>size_t</code>	Number of elements
<code>elemsize</code>	<code>size_t</code>	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.

Return `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.

See Also [“vec calloc” on page 439](#)
[“malloc” on page 421](#)
[“realloc” on page 427](#)

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.

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

Prototype `#include <stdlib.h>`
`div_t div(int numer, int denom);`

Parameters Parameters for this facility are:

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;
```

Return `div()` divides `denom` into `numer` and returns the quotient and remainder as a `div_t` type.

See Also [“fmod” on page 194](#)
[“ldiv” on page 419](#)
[“div_t” on page 81](#)

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.

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

Prototype `#include <stdlib.h>`
`void exit(int status);`

Parameters Parameters for this facility are:

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()`.

Return `exit()` does not return any value to the operating system. The status argument is kept to conform to the ANSI C Standard Library specification.

See Also [“abort” on page 401](#)
[“atexit” on page 403](#)

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;
}
```

free

Release previously allocated memory to heap.

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

Prototype	#include <stdlib.h> void free(void *ptr);	
Parameters	Parameters for this facility are: ptr void * A pointer to the allocated memory	
Remarks	The <code>free()</code> function releases a previously allocated memory block, pointed to by <code>ptr</code> , to the heap. The <code>ptr</code> argument should hold an address returned by the memory allocation functions <code>calloc()</code> , <code>malloc()</code> , or <code>realloc()</code> . Once the memory block pointed to by <code>ptr</code> has been released, it is no longer valid. The <code>ptr</code> variable should not be used to reference memory again until it is assigned a value from the memory allocation functions.	
See Also	“vec_free” on page 440 “calloc” on page 412 “malloc” on page 421 “realloc” on page 427	

Listing 35.9 For example of free() usage

Refer to [“Example of calloc\(\) usage.” on page 413](#).

getenv

Environment list access.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <stdlib.h> char *getenv(char *name);	
Parameters	Parameters for this facility are: name char * A buffer for the environment list	
Remarks	For Macintosh systems the <code>getenv()</code> is an empty function that always returns a null pointer (<code>NULL</code>). It is included in the Metrowerks <code>stdlib.h</code> header file to conform to the ANSI C Standard Library specification.	

Return `getenv()` returns NULL for the Mac. For Windows `getenv()` returns zero on failure or the environmental variable.

See Also [“system” on page 439](#)

Listing 35.10 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.

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

Prototype `#include <stdlib.h>`
`long int labs(long int j);`

Parameters Parameters for this facility are:

j long int The variable to be computed

Return `labs()` returns the absolute value of its argument as a value of type `long int`.

See Also [“fabs” on page 192](#)
[“abs” on page 402](#)

Listing 35.11 For example of labs() usage

Refer to ["Example of abs\(\) usage." on page 402](#).

ldiv

Compute the long integer quotient and remainder.

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

Prototype

```
#include <stdlib.h>
ldiv_t ldiv(long int numer, long int denom);
```

Parameters Parameters for this facility are:

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;
```

Return `ldiv()` divides `denom` into `numer` and returns the quotient and remainder as an `ldiv_t` type.

See Also ["fmod" on page 194](#)
["div" on page 414](#)
["ldiv_t" on page 81](#)
["lldiv_t" on page 82](#)

Listing 35.12 For example of ldiv() usage

Refer to ["Example of div\(\) usage." on page 414](#).

llabs

Compute long long integer absolute value.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <stdlib.h> long long llabs(long long j);
Parameters	Parameters for this facility are:
	j long long The variable to be computed
Return	llabs() returns the absolute value of its argument as a value of type long long.
See Also	"fabs" on page 192 "abs" on page 402

Listing 35.13 For example of llabs() usage

Refer to ["Example of abs\(\) usage." on page 402](#).

lldiv

Compute the long long integer quotient and remainder.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <stdlib.h> ldiv_t ldiv(long long numer, long long denom);
Parameters	Parameters for this facility are:
	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;
Return	lldiv() divides denom into numer and returns the quotient and remainder as an lldiv_t type.

See Also	“ldiv” on page 419 “fmod” on page 194 “lldiv_t” on page 82
----------	--

malloc

Allocate a block of heap memory.

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

Prototype

```
#include <stdlib.h>
void *malloc(size_t size);
```

Parameters Parameters for this facility are:

size size_t The size in bytes of the allocation

Remarks The `malloc()` function allocates a block of contiguous heap memory `size` bytes large.

Return `malloc()` returns a pointer to the first byte of the allocated block if it is successful and return a null pointer if it fails.

See Also	“vec_malloc” on page 441 “calloc” on page 412 “free” on page 416 “realloc” on page 427
----------	---

Listing 35.14 For example of malloc() usage

Refer to [“Example of calloc\(\) usage.” on page 413](#).

mblen

Compute the length of an encoded multibyte character, encoded as defined by the `LC_CTYPE` category of the current locale.

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

Prototype	#include <stdlib.h> int mblen(const char *s, size_t n);
Parameters	Parameters for this facility are: s const char * The multibyte array to measure n size_t The maximum size
Remarks	The <code>mblen()</code> function returns the length of the multibyte character pointed to by <code>s</code> . It examines a maximum of <code>n</code> characters.
Return	If <code>s</code> is a null pointer, the <code>mblen</code> function returns a nonzero or zero value signifying whether multibyte encoding do or do not have state-dependent encoding. If <code>s</code> is not a null pointer, the <code>mblen</code> function either returns 0 (if <code>s</code> points to the null character), or returns the number of bytes that are contained in the multibyte.
See Also	“Locale specification” on page 169 “mbtowc” on page 423

mbstowcs

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <stddlib.h> size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
Parameters	Parameters for this facility are: 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 <code>mbstowcs()</code> converts a sequence of multibyte characters encoded as defined by the <code>LC_CTYPE</code> category of the current locale from the character array pointed to by <code>s</code> and stores not more than <code>n</code> 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.

Return 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.

See Also [“Locale specification” on page 169](#)
[“wcstombs” on page 442](#)

mbtowc

Translate a multibyte character, encoded as defined by the `LC_CTYPE` category of the current locale, to a `wchar_t` type.

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

Prototype

```
#include <stdlib.h>
int mbtowc(wchar_t *pwc,
           const char *s, size_t n);
```

Parameters Parameters for this facility are:

<code>pwc</code>	<code>wchar_t *</code>	The wide character destination
<code>s</code>	<code>const char *s</code>	The string to convert
<code>n</code>	<code>size_t</code>	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 `wchar_t`, pointed to by `pwc`.

Return `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.

- See Also
- [“Locale specification” on page 169](#)
 - [“mblen” on page 421](#)
 - [“wctomb” on page 443](#)

_putenv

This functions lets you enter an argument into the environment list.

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

Prototype

```
#include <stdlib.h>
char *_putenv(const char *name);
```

Parameters Parameters for this facility are:

name const char * The item to add to the environment list

Return The function `_putenv()` returns NULL on success or minus one on failure to enter the environmental variable.

- See Also
- [“getenv” on page 417](#)
 - [“system” on page 439](#)

qsort

Sort an array.

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

Prototype

```
#include <stdlib.h>
void qsort(void *base, size_t nmemb,
```

```
size_t size,
int (*compare) (const void *, const void *))
```

Parameters	Parameters for this facility are:	
	base	void *
	nmemb	size_t
	size	size_t
	compare	void *

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.

See Also [“bsearch” on page 408](#)

Listing 35.15 For example of `qsort()` usage

Refer to [“Example of bsearch usage.” on page 409](#) .

rand

Generate a pseudo-random integer value.

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

Prototype `#include <stdlib.h>`

```
int rand(void);
```

Parameters None

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()`.

Return `rand()` returns a pseudo-random integer value between 0 and `RAND_MAX`.

See Also [“`srand`” on page 428](#)

Listing 35.16 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

realloc

Change the size of an allocated block of heap memory.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <stdlib.h> void *realloc(void *ptr, size_t size);
Parameters	Parameters for this facility are: ptr void * A pointer to an allocated block of memory size size_t The size of memory to reallocate
Remarks	The <code>realloc()</code> function changes the size of the memory block pointed to by <code>ptr</code> to <code>size</code> bytes. The <code>size</code> argument can have a value smaller or larger than the current size of the block <code>ptr</code> points to. The <code>ptr</code> argument should be a value assigned by the memory allocation functions <code>calloc()</code> and <code>malloc()</code> . If <code>size</code> is 0, the memory block pointed to by <code>ptr</code> is released. If <code>ptr</code> is a null pointer, <code>realloc()</code> allocates <code>size</code> 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.
Return	<code>realloc()</code> returns a pointer to the new block if it is successful and <code>size</code> is greater than 0. <code>realloc()</code> returns a null pointer if it fails or <code>size</code> is 0.
See Also	“vec_realloc” on page 441 “calloc” on page 412

[“free” on page 416](#)

[“malloc” on page 421](#)

Listing 35.17 For example of realloc() usage

Refer to [“Example of calloc\(\) usage.” on page 413](#).

srand

Sets the seed for the pseudo-random number generator.

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

Prototype

```
#include <stdlib.h>
void srand(unsigned int start);
```

Parameters Parameters for this facility are:

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.

See Also [“rand” on page 425](#)

Listing 35.18 For example of labs() usage

Refer to [“Example of rand\(\) usage.” on page 426](#).

strtod

Character array conversion to floating point value of type double.

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

Prototype

```
#include <stdlib.h>
double strtod( const char *nptr,
               char **endptr);
```

Parameters Parameters for this facility are:

	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 <code>strtod()</code> function converts a character array, pointed to by <code>nptr</code> , to a floating point value of type <code>double</code> . The character array can be in either decimal or hexadecimal floating point constant notation (e.g. <code>103.578</code> , <code>1.03578e+02</code> , or <code>0x1.9efef9p+6</code>).		
	If the <code>endptr</code> argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by <code>nptr</code> . This position marks the first character that is not convertible to a value of type <code>double</code> .		
	In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.		
	This function skips leading white space.		
Return	<code>strtod()</code> returns a floating point value of type <code>double</code> . If <code>nptr</code> cannot be converted to an expressible double value, <code>strtod()</code> returns <code>HUGE_VAL</code> , defined in <code>math.h</code> , and sets <code>errno</code> to <code>ERANGE</code> .		
See Also	"strtol" on page 431 "strtoul" on page 436 "errno" on page 83 "Integral type limits" on page 167 "Overview of math.h" on page 175 "scanf" on page 369		

Listing 35.19 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|
```

strtof

Character array conversion to floating point value of type float.

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

Prototype #include <stdlib.h>

```
float strtod( const char *nptr,
              char **endptr);
```

Parameters	Parameters for this facility are:	
	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 `double`.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

Return `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`.

See Also ["strtol" on page 431](#)
["strtod" on page 428](#)

strtol

Character array conversion to an integral value of type `long int`.

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

Prototype

```
#include <stdlib.h>
long int strtol(const char *nptr,
                char **endptr, int base);
```

Parameters	Parameters for this facility are:
------------	-----------------------------------

	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 <code>strtol()</code> function converts a character array, pointed to by <code>nptr</code> , expected to represent an integer expressed in radix <code>base</code> , to an integer value of type <code>long int</code> . A plus or minus sign (+ or -) prefixing the number string is optional.		
	The <code>base</code> argument in <code>strtol()</code> 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 <code>base</code> are permitted. If <code>base</code> is 0, then <code>strtol()</code> 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 <code>endptr</code> argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by <code>nptr</code> . This position marks the first character that is not convertible to a <code>long int</code> value.		
	In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.		
	This function skips leading white space.		
Return	<code>strtol()</code> returns an integer value of type <code>long int</code> . If the converted value is less than <code>LONG_MIN</code> , <code>strtol()</code> returns <code>LONG_MIN</code> and sets <code>errno</code> to <code>ERANGE</code> . If the converted value is greater than <code>LONG_MAX</code> , <code>strtol()</code> returns <code>LONG_MAX</code> and sets <code>errno</code> to <code>ERANGE</code> . The <code>LONG_MIN</code> and <code>LONG_MAX</code> macros are defined in <code>limits.h</code> .		
See Also	"strtod" on page 428 "strtoul" on page 436 "errno" on page 83 "Integral type limits" on page 167		

[“Overview of math.h” on page 175](#)

[“scanf” on page 369](#)

Listing 35.20 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.

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

Prototype

```
#include <stdlib.h>
long double strtold( const char *nptr,
                     char **endptr);
```

Parameters Parameters for this facility are:

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.

Return 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.

See Also ["strtol" on page 431](#)

["errno" on page 83](#)

["Overview of math.h" on page 175](#)

Listing 35.21 For example of `strtold()` usage

Refer to ["Example of `strtod\(\)` and `strtold\(\)` usage." on page 429.](#)

strtoll

Character array conversion to integer value of type long long int.

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

Prototype

```
#include <stdlib.h>
unsigned long int strtoul(const char *nptr,
    char **endptr, int base);
```

Parameters Parameters for this facility are:

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. A plus or minus sign (+ or -) prefixing the number string is optional.

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.

Return `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`

See Also [“`strtod`” on page 428](#)

[“`strtol`” on page 431](#)

[“`errno`” on page 83](#)

[“Integral type limits” on page 167](#)

[“Overview of `math.h`” on page 175](#)

Listing 35.22 For example of `strtoll()` usage

Refer to [“Example of `strtol\(\)`, `strtoul\(\)`, `strtoll\(\)`, and `strtoull\(\)` usage” on page 433](#).

strtoul

Character array conversion to integer value of type `unsigned long int`.

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

Prototype `#include <stdlib.h>`
`unsigned long int strtoul(const char *nptr,`
`char **endptr, int base);`

Parameters Parameters for this facility are:

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	<p>The <code>strtoul()</code> function converts a character array, pointed to by <code>nptr</code>, to an integer value of type <code>unsigned long int</code>, in base. A plus or minus sign prefix is ignored.</p> <p>The <code>base</code> argument in <code>strtoul()</code> 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 <code>base</code> are permitted. If <code>base</code> is 0, then <code>strtol()</code> and <code>strtoul()</code> convert the character array based on its format. Character arrays beginning with '<code>'0'</code>' are assumed to be octal, number strings beginning with '<code>'0x'</code>' or '<code>'0X'</code>' are assumed to be hexadecimal. All other number strings are assumed to be decimal.</p> <p>If the <code>endptr</code> argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by <code>nptr</code>. This position marks the first character that is not convertible to the functions' respective types.</p> <p>In other than the "<code>'C'</code>" locale, additional locale-specific subject sequence forms may be accepted.</p> <p>This function skips leading white space.</p>
Return	<p><code>strtoul()</code> returns an <code>unsigned integer value of type unsigned long int</code>. If the converted value is greater than <code>ULONG_MAX</code>, <code>strtoul()</code> returns <code>ULONG_MAX</code> and sets <code>errno</code> to <code>ERANGE</code>. The <code>ULONG_MAX</code> macro is defined in <code>limits.h</code></p>
See Also	<p>"strtod" on page 428 "strtol" on page 431 "errno" on page 83 "Integral type limits" on page 167 "Overview of math.h" on page 175</p>

Listing 35.23 For example of `strtoll()` usage

Refer to ["Example of `strtol\(\)`, `strtoul\(\)`, `strtoll\(\)`, and `strtoull\(\)` usage" on page 433](#).

strtoull

Character array conversion to integer value of type unsigned long long int.

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

Prototype

```
#include <stdlib.h>
unsigned long int strtoul(const char *nptr,
    char **endptr, int base);
```

Parameters Parameters for this facility are:

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 `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`. A plus or minus sign prefix is ignored.

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.

Return	strtoull() returns an unsigned integer value of type unsigned long long int. If the converted value is greater than ULONG_MAX, strtoull() returns ULONG_MAX and sets errno to ERANGE. The ULONG_MAX macro is defined in limits.h
See Also	"strtoul" on page 436 "strtol" on page 435

Listing 35.24 For example of strtol() usage

Refer to ["Example of strtol\(\), strtoul\(\), strtoll\(\), and strtoull\(\) usage" on page 433.](#)

system

Environment list assignment.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <stdlib.h> int system(const char *string);

NOTE The system() function is an empty function on MacOS t

Parameters Parameters for this facility are:

string const char * A OS system command

Return system() returns zero if successful or minus one on failure.

See Also ["getenv" on page 417](#)

vec_calloc

Clears and allocates memory on a 16 byte alignment.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	void * vec_calloc(size_t nmemb, size_t size);

Parameters	Parameters for this facility are:	
	nmemb	size_t Number of elements
	size	size_t The size of the elements
Remarks	The <code>vec_calloc()</code> function allocates contiguous space for <code>nmemb</code> elements of <code>size</code> . The space is initialized with zeroes.	
Return	<code>vec_calloc()</code> returns a pointer to the first byte of the memory area allocated. <code>vec_calloc()</code> returns a null pointer (<code>NULL</code>) if no space could be allocated.	
See Also	“calloc” on page 412 “vec_free” on page 440 “vec_malloc” on page 441 “vec_realloc” on page 441	

vec_free

Compatibility	Frees memory allocated by <code>vec_malloc</code> , <code>vec_calloc</code> and <code>vec_realloc</code>
Prototype	<code>void vec_free(void *ptr);</code>
Parameters	Parameters for this facility are:
	<code>ptr void *</code> A pointer to the allocated memory
Remarks	The <code>vec_free()</code> function releases a previously allocated memory block, pointed to by <code>ptr</code> , to the heap. The <code>ptr</code> argument should hold an address returned by the memory allocation functions <code>vec_calloc()</code> , <code>vec_malloc()</code> , or <code>vec_realloc()</code> . Once the memory block <code>ptr</code> points to has been released, it is no longer valid. The <code>ptr</code> variable should not be used to reference memory again until it is assigned a value from the memory allocation functions.
Return	There is no return.
See Also	“free” on page 416

[“vec_calloc” on page 439](#)

[“vec_malloc” on page 441](#)

[“vec_realloc” on page 441](#)

vec_malloc

Allocates memory on a 16 byte alignment.

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

Prototype `void * vec_malloc(size_t size);`

Parameters Parameters for this facility are:

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.

Return `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.

See Also [“malloc” on page 421](#)

[“vec_free” on page 440](#)

[“vec_calloc” on page 439](#)

[“vec_realloc” on page 441](#)

vec_realloc

Reallocates memory on a 16 byte alignment.

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

Prototype `void * vec_realloc(void * ptr, size_t size);`

Parameters Parameters for this facility are:

	ptr	void *	A pointer to an allocated block of memory
	size	size_t	The size of memory to reallocate
Return	<code>vec_realloc()</code> returns a pointer to the new block if it is successful and <code>size</code> is greater than 0. <code>realloc()</code> returns a null pointer if it fails or <code>size</code> is 0.		
See Also	“realloc” on page 427 “vec_free” on page 440 “vec_calloc” on page 439 “vec_malloc” on page 441		

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.

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

Prototype

```
#include <stdlib.h>
size_t wcstombs(char *s, const
                 wchar_t *pwcs, size_t n);
```

Parameters Parameters for this facility are:

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.

Return `wcstombs()` returns the number of bytes modified in the character array pointed to by `s`, not including a terminating null character, if any.

See Also [“Locale specification” on page 169](#), [“mbstowcs” on page 422](#)

wctomb

Translate a `wchar_t` type to a multibyte character encoded as defined by the `LC_CTYPE` category of the current locale.

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

Prototype

```
#include <stdlib.h>
int wctomb(char *s, wchar_t wchar);
```

Parameters Parameters for this facility are:

<code>s</code>	<code>char *</code>	A multibyte string buffer
<code>wchar</code>	<code>wchar_t</code>	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.

Return `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`.

See Also [“Locale specification” on page 169](#)
[“mbtowc” on page 423](#)

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 92](#), for a full description.
- For the function `itoa` see [“itoa” on page 95](#), for a full description.
- For the function `itow` see [“itow” on page 95](#), for a full description.
- For the function `ltoa` see [“ltoa” on page 96](#), for a full description.
- For the function `makepath` see [“makepath” on page 97](#), for a full description.
- For the function `splitpath` see [“splitpath” on page 99](#), for a full description.
- For the function `ultoa` see [“ultoa” on page 111](#), for a full description.
- For the function `wtoi` see [“wtoi” on page 120](#), 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 functions for comparing, copying, concatenating, and searching character arrays and arrays of larger items.

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 451](#) concatenates strings
- [“`strchr`” on page 452](#) searches by character
- [“`strcmp`” on page 453](#) compares strings
- [“`strcpy`” on page 456](#) copies strings
- [“`strcoll`” on page 454](#) compares string lexicographically
- [“`strcspn`” on page 457](#) find a substring in a string
- [“`strerror`” on page 458](#) retrieves an error message from an `errno` variable
- [“`strlen`” on page 459](#) returns string's length
- [“`strpbrk`” on page 464](#) look for an occurrence of a character from one string in another
- [“`strrchr`” on page 465](#) searches a string for a character
- [“`strspn`” on page 465](#) search for a character not in one string in another

- [“`strstr`” on page 467](#) searches a string for a string
- [“`strtok`” on page 468](#) retrieves the next token or substring
- [“`strxfrm`” on page 470](#) 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 460](#), concatenates strings with length specified.
- [“`strncmp`” on page 461](#) string compare with length specified
- [“`strncpy`” on page 462](#) 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 446](#) searches a memory block for a character
- [“`memcmp`” on page 448](#) compares a memory block
- [“`memcpy`” on page 449](#) copies a memory block
- [“`memmove`” on page 450](#) moves a memory block
- [“`memset`” on page 451](#) 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.

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

Prototype `#include <string.h>`
`void *memchr(const void *s, int c, size_t n);`

Parameters Parameters for this facility are:

<code>s</code>	<code>const void *</code>	The memory to search
<code>c</code>	<code>int</code>	The char to search for
<code>n</code>	<code>size_t</code>	The maximum length to search

Remarks	The <code>memchr()</code> function looks for the first occurrence of <code>c</code> in the first <code>n</code> characters of the memory area pointed to by <code>s</code> .
Return	<code>memchr()</code> returns a pointer to the found character, or a null pointer (<code>NULL</code>) if <code>c</code> cannot be found.
See Also	"strchr" on page 452 "strrchr" on page 465

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, lendest;

    // 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;
    lendest = 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.

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

Prototype `#include <string.h>`
`int memcmp(const void *s1,`
 `const void *s2, size_t n);`

Parameters Parameters for this facility are:

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.

Return 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.

See Also ["strcmp" on page 453](#)
["strncpy" on page 461](#)

Listing 36.2 For example of memcmp() usage

Refer to ["Example of memchr\(\) usage." on page 447](#).

memcpy

Copy a contiguous memory block.

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

Prototype

```
#include <string.h>
void *memcpy(void *dest,
             const void *source, size_t n);
```

Parameters Parameters for this facility are:

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.

Return memcpy() returns the value of dest.

See Also [“memmove” on page 450](#)

[“strcpy” on page 456](#)

[“strncpy” on page 462](#)

Listing 36.3 For example of memcpy() usage

Refer to [“Example of memchr\(\) usage.” on page 447](#).

memmove

Copy an overlapping contiguous memory block.

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

Prototype

```
#include <string.h>
void *memmove(void *dest,
              const void *source, size_t n);
```

Parameters Parameters for this facility are:

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.

Return `memmove()` returns the value of `dest`.

See Also [“memcpy” on page 449](#)

[“memset” on page 451](#)

[“strcpy” on page 456](#)

[“strncpy” on page 462](#)

Listing 36.4 For example of memmove() usage

Refer to [“Example of memchr\(\) usage.” on page 447](#).

memset

Set the contents of a block of memory to the value of a single character.

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

Prototype

```
#include <string.h>
void *memset(void *dest, int c, size_t n);
```

Parameters Parameters for this facility are:

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`.

Return `memset()` returns the value of `dest`.

Listing 36.5 For example of `memset()` usage

Refer to ["Example of `memchr\(\)` usage."](#) on page 447 .

strcat

Concatenate two character arrays.

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

Prototype

```
#include <string.h>
char *strcat(char *dest, const char *source);
```

Parameters Parameters for this facility are:

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.

Return `strcat()` returns the value of dest.

See Also [“`strcpy`” on page 456](#)

Listing 36.6 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.

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

Prototype `#include <string.h>`
`char *strchr(const char *s, int c);`

Parameters Parameters for this facility are:

s	const char *	The string to search
c	int	The char to search for

Remarks	The <code>strchr()</code> function searches for the first occurrence of the character <code>c</code> in the character array pointed to by <code>s</code> . The <code>s</code> argument must point to a null terminated character array.
Return	<code>strchr()</code> returns a pointer to the successfully located character. If it fails, <code>strchr()</code> returns a null pointer (<code>NULL</code>).
See Also	“memchr” on page 446 “strchr” on page 465

Listing 36.7 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.

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

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

Parameters Parameters for this facility are:

<code>s1</code>	<code>const char *</code>	The string to compare
<code>s2</code>	<code>const char *</code>	The comparison string

Remarks	The <code>strcmp()</code> function compares the character array pointed to by <code>s1</code> to the character array pointed to by <code>s2</code> . Both <code>s1</code> and <code>s2</code> must point to null terminated character arrays.
Return	<code>strcmp()</code> returns a zero if <code>s1</code> and <code>s2</code> are equal, a negative value if <code>s1</code> is less than <code>s2</code> , and a positive value if <code>s1</code> is greater than <code>s2</code> .
See Also	“memcmp” on page 448 “strcoll” on page 454 “strncmp” on page 461

Listing 36.8 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.

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

Prototype	#include <string.h> int strcoll(const char *s1, const char *s2);	
Parameters	Parameters for this facility are: s1 const char * The string to compare s2 const char * The comparison string	
Remarks	The <code>strcoll()</code> function compares two character arrays based on the <code>LC_COLLATE</code> component of the current locale. The Metrowerks C implementation of <code>strcoll()</code> compares two character arrays using <code>strcmp()</code> . It is included in the string library to conform to the ANSI C Standard Library specification.	
Return	<code>strcoll()</code> returns zero if <code>s1</code> is equal to <code>s2</code> , a negative value if <code>s1</code> is less than <code>s2</code> , and a positive value if <code>s1</code> is greater than <code>s2</code> .	
See Also	“Locale specification” on page 169 “memcmp” on page 448 “strcmp” on page 453 “strncmp” on page 461,	

Listing 36.9 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 < 0)
        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.

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

Prototype `#include <string.h>
char *strcpy(char *dest, const char *source);`

Parameters Parameters for this facility are:

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.

Return `strcpy()` returns the value of `dest`.

See Also [“memcpy” on page 449](#)
[“memmove” on page 450](#)
[“strncpy” on page 462](#)

Listing 36.10 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.

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

Prototype `#include <string.h>
size_t strcspn(const char *s1, const char *s2);`

Parameters Parameters for this facility are:

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`.

Return `strcspn()` returns the index of the first character in `s1` that matches a character in `s2`.

See Also [“strpbrk” on page 464](#)
[“strspn” on page 465](#)

Listing 36.11 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.

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

Prototype `#include <string.h>`
`char *strerror(int errnum);`

Parameters Parameters for this facility are:

`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

Return `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`.

Listing 36.12 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
```

strlen

Compute the length of a character array.

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

Prototype `#include <string.h>`
`size_t strlen(const char *s);`

Parameters Parameters for this facility are:

`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.

Return `strlen()` returns the number of characters in a character array not including the terminating null character.

Listing 36.13 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.

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

Prototype `#include <string.h>`
`char *strncat(char *dest,`
 `const char *source, size_t n);`

Parameters Parameters for this facility are:

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`').

Return `strncat()` returns the value of `dest`.

See Also [“**strcat**” on page 451](#)

Listing 36.14 Example of **strncat() usage.**

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

int main(void)
{
    static char s1[100] = "abcdefghijklmнопqrstuvwxyz";
    static char s2[] = "wxyz0123456789";

    strncat(s1, s2, 4);
    puts(s1);

    return 0;
}
Output:
abcdefghijklmнопqrstuvwxyzwxyz
```

strncmp

Compare a specified number of characters.

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

Prototype `#include <string.h>
int strncmp(const char *s1,
 const char *s2, size_t n);`

Parameters Parameters for this facility are:

<code>s1</code>	<code>const char *</code>	The string to compare
<code>s2</code>	<code>const char *</code>	The comparison string
<code>n</code>	<code>size_t</code>	The maximum number of characters to compare

Remarks The `strncmp()` 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.

Return `strncmp()` 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*.

See Also [“memcmp” on page 448](#)
[“strcmp” on page 453](#)

Listing 36.15 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.

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

Prototype `#include <string.h>`
`char *strncpy(char *dest,`
`const char *source, size_t n);`

Parameters Parameters for this facility are:

	dest	char *	The destination string
	source	const char *	The source to copy
	n	size_t	The maximum length to copy
Remarks	The <code>strncpy()</code> function copies a maximum of <code>n</code> characters from the character array pointed to by <code>source</code> to the character array pointed to by <code>dest</code> . Neither <code>dest</code> nor <code>source</code> need necessarily point to null terminated character arrays. Also, <code>dest</code> and <code>source</code> must not overlap.		
	If a null character (' \0 ') is reached in <code>source</code> before <code>n</code> characters have been copied, <code>strncpy()</code> continues padding <code>dest</code> with null characters until <code>n</code> characters have been added to <code>dest</code> .		
	The function does not terminate <code>dest</code> with a null character if <code>n</code> characters are copied from <code>source</code> before reaching a null character.		
Return	<code>strncpy()</code> returns the value of <code>dest</code> .		
See Also	“<code>memcpy</code>” on page 449 “<code>memmove</code>” on page 450 “<code>strcpy</code>” on page 456		

Listing 36.16 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.

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

Prototype `#include <string.h>
char *strpbrk(const char *s1, const char *s2);`

Parameters Parameters for this facility are:

`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.

Return `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.

See Also [“strcspn” on page 457](#)

Listing 36.17 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.

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

Prototype

```
#include <string.h>
char *strrchr(const char *s, int c);
```

Parameters Parameters for this facility are:

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.

Return `strrchr()` returns a pointer to the character found or returns a null pointer (`NULL`) if it fails.

See Also ["memchr" on page 446](#)

["strchr" on page 452](#)

Listing 36.18 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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <string.h> size_t strspn(const char *s1, const char *s2);
Parameters	Parameters for this facility are: s1 const char * The string to search s2 const char * A list of characters to look for
Remarks	The <code>strspn()</code> function finds the first character in the null terminated character string <code>s1</code> that is not in the null terminated string <code>s2</code> . The function starts examining characters at the beginning of <code>s1</code> and continues searching until a character in <code>s1</code> does not match any character in <code>s2</code> . Both <code>s1</code> and <code>s2</code> must point to null terminated character arrays.
Return	<code>strspn()</code> returns the index of the first character in <code>s1</code> that does not match a character in <code>s2</code> .
See Also	“strpbrk” on page 464 “strcspn” on page 457

Listing 36.19 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.

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

Prototype

```
#include <string.h>
char *strstr(const char *s1, const char *s2);
```

Parameters Parameters for this facility are:

 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.

Return `strstr()` returns a pointer to the first occurrence of `s2` in `s1` and returns a null pointer (`NULL`) if `s2` cannot be found.

See Also ["memchr" on page 446](#)

["strchr" on page 452](#)

Listing 36.20 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.

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

Prototype

```
#include <string.h>
char *strtok(char *str, const char *sep);
```

Parameters Parameters for this facility are:

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.

NOTE The sep argument can be different for each call to `strtok()`.

NOTE `strtok()` modifies the character array pointed to by str.

Return 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.

Listing 36.21 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.

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

Prototype

```
#include <string.h>
size_t strxfrm(char *dest,
                const char *source, size_t n);
```

Parameters Parameters for this facility are:

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.

Return `strxfrm()` returns the length of `dest` after it has received `source`.

See Also [“Locale specification” on page 169](#)
[“strcpy” on page 456](#)

Listing 36.22 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 102](#), for a full description.
- For the function `stricmp` see “[stricmp” on page 102](#), for a full description.
- For the function `strlwr` see “[strlwr” on page 103](#), for a full description.
- For the function `strnicmp` see “[strnicmp” on page 106](#), for a full description.
- For the function `strnset` see “[strnset” on page 108](#), for a full description.
- For the function `strrev` see “[strrev” on page 108](#), for a full description.
- For the function `strset` see “[strset” on page 109](#), for a full description.
- For the function `strupr` see “[strupr” on page 110](#), for a full description.

- For the function `wcsdup` see “[wcsdup](#)” on page 113, for a full description.
- For the function `wcsicmp` see “[wcsicmp](#)” on page 113, for a full description.
- For the function `wcslwr` see “[wcslwr](#)” on page 114, for a full description.
- For the function `wcsncmp` see “[wcsncmp](#)” on page 116, for a full description.
- For the function `wcsnset` see “[wcsnset](#)” on page 117, for a full description.
- For the function `wcsrev` see “[wcsrev](#)” on page 118, for a full description.
- For the function `wcsset` see “[wcsset](#)” on page 118, for a full description.
- For the function `wcsspnp` see “[wcsspnp](#)” on page 119, for a full description.
- For the function `wcsupr` see “[wcsupr](#)” on page 119, for a full description.
- For the function `wtoi` see “[wtoi](#)” on page 120, for a full description.

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 473](#)”.

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.

Table 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

Function	Type-Generic Macro	Function	Type-Generic Macro
expm1	expm1	fabs	fabs
fdim	fdim	floor	floor
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 476](#) is a structure for storing time data.
- [“`tzname`” on page 478](#) an array that stores the time zone abbreviations
- [“`asctime`” on page 478](#) to convert a `tm` structure type to a char array
- [“`clock`” on page 479](#) to determine the relative time since the system was started
- [“`ctime`” on page 481](#) to convert a `time_t` type to a char array
- [“`difftime`” on page 481](#) to determine the difference between two times
- [“`gmtime`” on page 483](#) to determine Greenwich Mean Time
- [“`localtime`” on page 484](#) to determine the local time
- [“`mktime`” on page 484](#) to convert a `tm` structure to `time_t` type
- [“`strftime`” on page 485](#) to format time as a C string
- [“`time`” on page 491](#) to determine a number of seconds from a set time
- [“`tzset`” on page 492](#) 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.

NOTE 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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <time.h> <pre>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; };</pre>
Remarks	The <code>tm</code> structure members are listed “Tm Structure Members.” on page 477 .
NOTE	The <code>tm_isdst</code> flag is positive if Daylight Savings Time is in effect, zero if it is not, and negative if such information is not available.

Table 38.1 Tm Structure Members.

Field	Description	Range min - max
<code>int tm_sec</code>	Seconds after the minute	0 - 59
<code>int tm_min</code>	Minutes after the hour	0 - 59
<code>int tm_hour</code>	Hours after midnight	0 - 23
<code>int tm_mday</code>	Day of the month	1 - 31
<code>int tm_mon</code>	Months after January	0 - 11
<code>int tm_year</code>	Up to 136 years after 1900	1900 - 2036
<code>int tm_wday</code>	Days after Sunday	0 - 6
<code>int tm_yday</code>	Days after January 1	0 - 365
<code>int tm_isdst</code>	Daylight Savings Time flag	

tzname

The _tzname_ array contains the names (abbreviations) of the time zones for local standard time and DST.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <time.h> extern char *tzname [2] ;
See Also	“tzset” on page 492

asctime

Convert a tm structure to a character array.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <time.h> char *asctime(const struct tm *timeptr);
Parameters	Parameters for this facility are: 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.
Return	asctime() returns a null terminated character array pointer containing the converted tm structure.
See Also	“ctime” on page 481 “strftime” on page 485

Listing 38.1 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

clock

A program relative invocation of the system time.

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

Prototype `#include <time.h>`
`clock_t clock(void);`

Parameters None

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`.

NOTE The programmer should be aware that `clock_t`, defined in `time.h`, has a finite size that varies depending upon the target system.

Return 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.

Listing 38.2 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.

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

Prototype `#include <time.h>`
`char *ctime(const time_t *timer);`

Parameters Parameters for this facility are:

`timer const time_t *` The address of the `time_t` variable

Remarks The `ctime()` function converts a `time_t` type to a character array with the same format as generated by `asctime()`.

Return `ctime()` returns a null terminated character array pointer containing the converted `time_t` value.

See Also [“asctime” on page 478](#)

[“strftime” on page 485](#)

Listing 38.3 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
```

difftime

Compute the difference between two `time_t` types.

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

Prototype `#include <time.h>
double difftime(time_t t1, time_t t2);`

Parameters Parameters for this facility are:

t1	time_t	A time_t variable to compare
t2	time_t	A time_t variable to compare

Return `difftime()` returns the difference of `t1` minus `t2` expressed in seconds.

Listing 38.4 Example of difftime usage.

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

int main(void)
{
    time_t t1, t2;
    struct tm *currtime;
    double midnight;

    time(&t1);
    currtime = localtime(&t1);

    currtime->tm_sec = 0;
    currtime->tm_min = 0;
    currtime->tm_hour = 0;
    currtime->tm_mday++;

    t2 = mktime(currtime);

    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.

Compatibility

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

Prototype

```
#include <time.h>
struct tm *gmtime(const time_t *timer);
```

Parameters

Parameters for this facility are:

timer const `time_t` * The address of the `time_t` variable

Remarks

The `gmtime` function converts the calendar time pointed to by `timer` into a broken-down time, expressed as UTC.

Return

The `gmtime()` function returns a pointer to that object.

Listing 38.5 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

localtime

Convert a `time_t` type to a `struct tm` type.

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

Prototype

```
#include <time.h>
struct tm *localtime(const time_t *timer);
```

Parameters Parameters for this facility are:

`timer` `const time_t *` The address of the `time_t` 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.

Return `localtime()` converts `timer` and returns a pointer to a `struct tm`.

See Also [“mktime” on page 484](#)

For Usage Refer to the example for [“Example of difftime usage.” on page 482](#).

mktime

Convert a `struct tm` item to a `time_t` type.

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

Prototype

```
#include <time.h>
time_t mktime(struct tm *timeptr);
```

Parameters Parameters for this facility are:

`timeptr` `struct tm *` The address of the `tm` structure

Remarks The `mktime()` 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`.

Return `mktime()` returns the converted `tm` structure as a `time_t` type.

See Also [“localtime” on page 484](#)

Listing 38.6 For example of usage

Refer to the example for [“Example of difftime usage.” on page 482](#).

strftime

Format a `tm` structure.

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

Prototype `#include <time.h>`
`size_t strftime(char *s, size_t maxsize,`
`const char *format,`
`const struct tm *timeptr);`

Parameters Parameters for this facility are:

<code>s</code>	<code>char *</code>	A string to hold the formatted time
<code>maxsize</code>	<code>size_t</code>	Max length of formatted string
<code>format</code>	<code>const char *</code>	The format string
<code>timeptr</code>	<code>const struct tm*</code>	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.

NOTE Refer to "[Strftime\(\) conversion characters](#)" on page 486 for a list of format specifiers.

Table 38.2 **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"

Char	Description
e	The day of the month as a decimal number; a single digit is preceded by a space.
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.

Char	Description
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"
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.

Return 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.

Listing 38.7 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);
```

time.h*Date and time*

```
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,\\
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,\\
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);

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.

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

Prototype `#include <time.h>`
 `time_t time(time_t *timer);`

Parameters Parameters for this facility are:

 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.

Return `time()` returns the system current calendar time.

Listing 38.8 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.

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

Prototype `#include <time.h>`
 `void tzset(void);`

Parameters None

Remarks The function `tzset()` reads the value of the “TZ” environment variable and internalizes it into the time zone functionality of the program.

See Also [“tzname” on page 478](#)

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 101](#), for a full description.

time.h

Non Standard <time.h> Functions

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 Macintosh 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:

- [“chdir” on page 497](#) change the directory
- [“close” on page 499](#) close a file opened with open
- [“cuserid” on page 502](#) retrieves the current user’s ID
- [“cwait” on page 503](#) Wait for a process to end.
- [“dup” on page 503](#) duplicates a file handle
- [“dup2” on page 504](#) duplicates to an existing file handle
- [“exec functions” on page 505](#) executes programs from within a program
- [“getcwd” on page 507](#) gets the current working directory
- [“getlogin” on page 508](#) returns a login name
- [“getpid” on page 508](#) returns the process ID
- [“isatty” on page 510](#) determines if a file ID is attached to a terminal
- [“lseek” on page 511](#) seek when opened with open
- [“read” on page 512](#) read when opened with open

- [“rmdir” on page 513](#) removes a directory or folder
- [“sleep” on page 515](#) pauses a program
- [“spawn functions” on page 517](#) spawns a child process
- [“ttynname” on page 518](#) determines a terminal id
- [“unlink” on page 519](#) deletes a file
- [“write” on page 520](#) 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” on page 27](#), for information on POSIX naming conventions.

Table 39.1 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

access

Determines the files accessibility

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

Prototype

```
#include <unistd.h>
int access(const char *fname, int mode);
```

Parameters Parameters for this facility are:

	fname	const char *	The file to check
	mode	int	The file mode to test for
Remarks	The “ Access Test Modes ” on page 496 lists the file modes that may be tested.		
Return	If the access is allowed then zero is returned. A negative number is returned if the access is not allowed.		
See Also	“creat” on page 124 “open” on page 126 “close” on page 499		

chdir

Change the current directory.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <unistd.h> int chdir(const char *path); int _chdir(const char *path);
Parameters	Parameters for this facility are:
	path char * The new pathname
Remarks	The function <code>chdir()</code> 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 497
Return	<code>chdir()</code> returns zero, if successful. If unsuccessful <code>chdir()</code> returns negative one and sets <code>errno</code> .
See Also	“Overview of errno.h” on page 83

Listing 39.1 Example of chdir() usage.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stat.h>
```

unistd.h

Overview of unistd.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 514
```

close

Close an open file.

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

Prototype `#include <unistd.h>`
`int close(int fildes);`

Parameters Parameters for this facility are:

	fildes int	The file descriptor
Remarks	The <code>close()</code> function closes the file specified by the argument. This argument is the value returned by <code>open()</code> . Example of usage is given in " Example of close() usage. " on page 500	
Return	If successful, <code>close()</code> returns zero. If unsuccessful, <code>close()</code> returns negative one and sets <code>errno</code> .	
See Also	"open" on page 126 "fclose" on page 303 "errno" on page 83	

Listing 39.2 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\nnone 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)
{
    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.

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

Prototype `#include <unistd.h>`
`char *cuserid(char *string);`

Parameters Parameters for this facility are:

`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 503](#)

NOTE For the MacOS, the login name is returned.

Return `cuserid()` returns a character pointer to the current user's ID.

NOTE For the MacOS, the users name is set using the sharing control panel

Listing 39.3 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

Compatibility Windows compatible function.

Prototype `#include <unistd.h>`
`int cwait(int *termstat, int pid, int action);`
`int _cwait(int *termstat, int pid, int action);`

Parameters Parameters for this facility are:

termstat	int	The termination status
pid	int	Process ID
action	int	The action is ignored

Return .The process exit code is returned.

See Also [“exec functions” on page 505](#)

[“spawn functions” on page 517](#)

dup

Duplicates a file handle.

Compatibility	Windows compatible header yet may also be implemented in other headers.
Prototype	#include <io.h> int dup(int _a) int _dup(int _a)
Parameters	Parameters for this facility are: _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.
Return	Anew file handle is returned upon success or a negative one otherwise.
See Also	“dup2” on page 504

dup2

Duplicate a file handle unto an existing handle.

Compatibility	Windows compatible header yet may also be implemented in other headers.
Prototype	#include <io.h> int dup2(int _a, int _b int _dup2(int _a, int _b
Parameters	Parameters for this facility are: _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.
Return	Zero is returned upon success and a negative one upon failure.
See Also	“dup” on page 503

exec functions

Load and execute a child process within the current program memory.

Compatibility

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

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

exec

Prototype `#include <unistd.h>`
`int exec(const char *path, ...);`
`int _exec(const char *path, ...);`

execl

Prototype `int execl(const char *path, ...);`
`int _execl(const char *path, ...);`

execle

Prototype `int execle(const char *path, ...);`
`int _execle(const char *path, ...);`

execlp

Prototype `int execlp(const char *path, ...);`
`int _execlp(const char *path, ...);`

execv

Prototype `int execv(const char *path, ...);`
`int _execv(const char *path, ...);`

execve

```
Prototype int execve(const char *path, ...);  
          int _execve(const char *path, ...);
```

excevp

```
Prototype    int excevp(const char *path, ...);  
             int excevp(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

Parameters **Parameters for this facility are:**

Launches the application named and then quits upon successful launch. Example of usage is given in [“Example of exec\(\) usage.” on page 506](#).

Returns If successful `exec()` returns zero. If unsuccessful `exec()` returns negative one and sets `errno` according to the error.

NOTE 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.

See Also [“Overview of SIOUX” on page 251](#)
 [“Overview of errno.h” on page 83](#)

Listing 39.4 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.

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

Prototype `#include <unistd.h>`
`char *getcwd(char *buf, int size);`
`char *_getcwd(char *buf, int size);`

Parameters Parameters for this facility are:

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.

Return If successful, `getcwd()` returns a pointer to the buffer. If unsuccessful, `getcwd()` returns NULL and sets `errno`.

See Also [“Overview of `errno.h`” on page 83](#)

Listing 39.5 For example of getcwd usageRefer to ["Example of chdir\(\) usage." on page 497](#).**getlogin**

Retrieve the username that started the process.

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

Prototype `#include <unistd.h>`
`char *getlogin(void);`

Parameters None

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 508](#)

NOTE The Mac doesn't have a login, so this function returns the Owner Name from the File Sharing Setup Control Panel

Return `getlogin()` returns a character pointer.

Listing 39.6 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.

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

Prototype `#include <unistd.h>`

Table 39.2 getpid() Macros

Macro	Represents
<code>#define getpid()</code>	Process ID
<code>#define getppid()</code>	Parent process ID
<code>#define getuid()</code>	Real user ID
<code>#define geteuid()</code>	Effective user ID
<code>#define getgid()</code>	Real group ID
<code>#define getegid()</code>	Effective group ID
<code>#define getpgrp()</code>	Process group ID

Parameters None

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 509](#)

Return `getpid()` returns an integer value.

NOTE 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.

Return `getpid()` always returns a value. There is no error returned.

Listing 39.7 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

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

Prototype `#include <unistd.h>`
`int isatty(int fildes);`
`int _isatty(int fildes);`

Parameters Parameters for this facility are:

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\(\) ttynname\(\) usage.” on page 510](#)

Return `isatty()` returns a non-zero value if the file is attached to the console. It returns zero if Input/Output redirection is in effect.

See Also [“ccommand” on page 50](#)

Listing 39.8 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
```

lseek

Seek a position on a file stream.

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

Prototype `#include <unistd.h>`
`long lseek(int fildes, long offset, int origin);`

Parameters Parameters for this facility are:

<code>fildes</code>	<code>int</code>	The file descriptor
<code>offset</code>	<code>long</code>	The offset to move in bytes
<code>origin</code>	<code>int</code>	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 512.](#)

Table 39.3 The lseek offset positions.

Macro	Meaning
SEEK_SET	Beginning of file
SEEK_CUR	Current Position
SEEK_END	End of File

Return 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.

See Also
[“fseek” on page 340](#)
[“ftell” on page 343](#)
[“open” on page 126](#)

Listing 39.9 For example of lseek() usage

Refer to [“Example of close\(\) usage.” on page 500.](#)

read

Read from a file stream that has been opened in binary mode for unformatted Input/Output.

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

Prototype `#include <unistd.h>`
`int read(int fildes, char *buf, int count);`

Parameters Parameters for this facility are:

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 <code>read()</code> copies the number of bytes specified by the <code>count</code> 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 <code>unistd.h:write()</code> , and <code>fcntl.h:open()</code> only.
Return	<code>read()</code> returns the number of bytes actually read from the file. In case of an error a value of negative one is returned and <code>errno</code> is set.
See Also	“fread” on page 330 “open” on page 126

Listing 39.10 For example of read() usage

Refer to [“Example of close\(\) usage.” on page 500](#).

rmdir

Delete a directory or folder.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <unistd.h> int rmdir(const char *path);</pre>
Parameters	Parameters for this facility are:
	path const char * The pathname of the directory being removed
Remarks	The <code>rmdir()</code> function removes the directory (folder) specified by the argument.
Return	If successful, <code>rmdir()</code> returns zero. If unsuccessful, <code>rmdir()</code> returns negative one and sets <code>errno</code> .
See Also	“mkdir” on page 272 “errno” on page 83

Listing 39.11 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);  
}  
  
/* 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 497.
```

sleep

Delay program execution for a specified number of seconds.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <unistd.h> unsigned int sleep(unsigned int sleep);
Parameters	Parameters for this facility are: sleep unsigned int The length of time in seconds
Remarks	The function <code>sleep()</code> 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 516
Return	<code>sleep()</code> returns zero.

Listing 39.12 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

NOTE These functions are currently Windows only

spawnl

Prototype `int spawnl(int,const char *, ...);
int _spawnl(int,const char *, ...);`

spawnv

Prototype `int spawnv(int,const char *,const char *const*);
int _spawnv(int,const char *,const char *const*);`

spawnle

Prototype `int spawnle(int,const char *,...);
int _spawnle(int,const char *,...);`

spawnve

Prototype `int spawnve(int,const char *,const char *const*,
const char *const*);
int _spawnve(int,const char *,const char *const*,
const char *const*);`

spawnlp

Prototype `int spawnlp(int,const char *,...);
int _spawnlp(int,const char *,...);`

spawnvp

Prototype `int spawnvp(int,const char *,const char *const *);`

```
int _spawnvp(int, const char *, const char *const *);
```

spawnlpe

Prototype `int spawnlpe(int, const char *, ...);
int _spawnlpe(int, const char *, ...);`

spawnvpe

Prototype `int spawnvpe(int, const char *, const char *const *,
const char *const*);
int _spawnvpe(int, const char *, const char *const *,
const char *const*);`

Parameters All `spawn` functions take a variable argument list as a parameter.

Return The child process's exit status is returned.

See Also [“exec functions” on page 505](#)

ttyname

Retrieve the name of the terminal associated with a file ID.

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

Prototype `#include <unistd.h>
char *ttyname(int fildes);`

Parameters Parameters for this facility are:

 fildes int The file descriptor

Remarks The function `ttyname()` retrieves the name of the terminal associated with the file ID.

Return `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.

Listing 39.13 For example of `ttyname()` usage

Refer to [“Example of `isatty\(\)` `ttyname\(\)` usage.” on page 510](#).

unlink

Delete (unlink) a file.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <unistd.h> int unlink(const char *path);
Parameters	Parameters for this facility are:
	path const char * A pathname of the file to remove
Remarks	The function <code>unlink()</code> removes the specified file from the directory. Example of usage is given in " "Example of <code>unlink()</code> usage." on page 519
Return	If successful, <code>unlink()</code> returns zero. If unsuccessful, it returns a negative one.
See Also	"rmdir" on page 513 "mkdir" on page 272

Listing 39.14 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.

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

Prototype `#include <unistd.h>`
`int write(int fildes, const char* buf,`
`size_t count)`

Parameters Parameters for this facility are:

	<code>fildes</code>	<code>int</code>	The file descriptor
	<code>buf</code>	<code>const char *</code>	The address of the buffer being written
	<code>count</code>	<code>size_t</code>	The size of the buffer being written
Remarks	The function <code>write()</code> copies the number of bytes in the <code>count</code> argument from the character array buffer to the file <code>fildes</code> . The file position is then incremented by the number of bytes written.		
NOTE	This function should be used in conjunction with " "read" on page 512 , and " "open" on page 126 only.		
Return	<code>write()</code> returns the number of bytes actually written.		
See Also	"fwrite" on page 346 "read" on page 512 "open" on page 126		

Listing 39.15 For example of write() usage

Refer to "[Example of close\(\) usage." on page 500](#).

unistd.h

Overview of unistd.h

unix.h

The header file `unix.h` contains two global variables that are useful for porting a program from UNIX to Macintosh.

Overview of unix.h

The header file `unix.h` contains two global variables that are useful for porting a program from UNIX. .

The globals variables in `unix.h` are:

- “[_fcreator](#)” on page 523 sets a Macintosh file creator
- “[_ftype](#)” on page 524 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.

Compatibility This global identifier is Macintosh Only

Prototype `#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 524](#) is an example of the use of the global variable _fcreator.

_ftype

To specify a Macintosh file type.

Compatibility This global identifier is Macintosh Only

Prototype `#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 524](#) is an example of the use of the global variable _ftype.

TIP The value assigned to _fcreate and _ftype is a ResType (i.e. four character constant).

Listing 40.1 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 = 'ttxt';
    _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 527](#) to set a file modification time
- [“utimes” on page 530](#) 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” on page 27](#), for information on POSIX naming conventions.

utime

Sets a file’s modification time.

utime.hOverview of *utime.h*

Compatibility	This function is implemented on Macintosh and Windows only. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	<pre>#include <utime.h> int utime(const char *path, /* Mac */ const struct utimbuf *buf); int utime(const char *path, /* Windows */ struct utimbuf *buf);</pre>	
Parameters	Parameters for this facility are:	
	path const char *	The pathname as a string
	buf const struct utimbuf *	The address of a struct that will hold a file's time information
Remarks	<p>This function sets the modification time for the file specified in <code>path</code>. Since the Macintosh does not have anything that corresponds to a file's access time, it ignores the <code>actime</code> field in the <code>utimbuf</code> structure.</p> <p>If <code>buf</code> is <code>NULL</code>, <code>utime()</code> sets the modification time to the current time. If <code>buf</code> points to a <code>utimbuf</code> structure, <code>utime()</code> sets the modification time to the time specified in the <code>modtime</code> field of the structure.</p>	
	The <code>utimbuf</code> structures contains the fields in Table 41.2 .	

Table 41.1 The `utimbuf` structure

This field...	is the...
<code>time_t</code>	<code>actime</code> Access time for the file. Since the Macintosh has nothing that corresponds to this, <code>utime()</code> ignores this field.
<code>time_t</code>	<code>modtime</code> The last time this file was modified.

Return If it is successful, `utime()` returns zero. If it encounters an error, `utime()` returns `-1` and sets `errno`.

See Also [“utimes” on page 530](#)
[“ctime” on page 481](#)

[“mktime” on page 484](#)

[“stat” on page 273](#)

[“fstat” on page 271](#)

Listing 41.1 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

Compatibility This function is implemented on Macintosh and Windows only.

Please refer to <http://www.metrowerks.com/docs/MSLCompatibility> for a Compatibility list.

Prototype `#include <utime.h>
int utimes(const char *path,
 struct timeval buf[2]);`

Parameters Parameters for this facility are:

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 [Table 41.2](#).

Table 41.2 The timeval structure

This field	is the
int t	tv_sec
int	tv_usec

The first element of buf is the access time.

NOTE Since the Macintosh does not have anything that corresponds to a file's access time, it ignores that element of the array.

Return If it is successful, `utimes()` returns 0. If it encounters an error, `utimes()` returns -1 and sets `errno`.

See Also

[“utime” on page 527](#)
[“ctime” on page 481](#)
[“mktime” on page 484](#)
[“fstat” on page 271](#)
[“stat” on page 273](#)

Listing 41.2 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 */

    /* Set the times for the two entries in buf */
    buf[0].tv_sec = ...; /* Set to date */
    buf[0].tv_usec = ...; /* Set to zero */
    buf[1].tv_sec = ...; /* Set to date */
    buf[1].tv_usec = ...; /* Set to zero */
}
```

utime.h

Overview of utime.h

```
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

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 533](#) 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” on page 27](#), for information on POSIX naming conventions.

uname

Gets information about the system you are using.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <utsname.h> int uname(struct utsname *name);
Parameters	Parameters for this facility are: 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 Table 42.1 . All the fields are null-terminated strings.

Table 42.1 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.

Return If it is successful, uname() returns zero. If it encounters an error, uname() returns -1 and sets errno.

See Also [“fstat” on page 271](#)
[“stat” on page 273](#)

Listing 42.1 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 543](#) behaves like `fgetc` for wide character arrays
- [“fgetws” on page 544](#) behaves like `fgets` for wide character arrays
- [“fputwc” on page 545](#) behaves like `fputc` for wide character arrays
- [“fputws” on page 545](#) behaves like `fputs` for wide character arrays
- [“fwprintf” on page 546](#) behaves like `fprintf` for wide character arrays
- [“fwscanf” on page 547](#) behaves like `scanf` for wide character arrays
- [“getwc” on page 548](#) behaves like `getc` for wide character arrays
- [“getwchar” on page 548](#) behaves like `getchar` for wide character arrays
- [“putwc” on page 552](#) behaves like `putc` for wide character arrays
- [“putwchar” on page 553](#) behaves like `putchar` for wide character arrays
- [“swprintf” on page 553](#) behaves like `sprintf` for wide character arrays

- [“swscanf” on page 554](#) behaves like `sscanf` for wide character arrays
- [“wprintf” on page 586](#) behaves like `printf` for wide character arrays
- [“wscanf” on page 590](#) behaves like `scanf` for wide character arrays
- [“vfwprintf” on page 558](#), behaves like `vfprintf` for wide character arrays
- [“vfwscanf” on page 555](#) behaves like `vfscanf` for wide character arrays
- [“vwscanf” on page 556](#) behaves like `vsscanf` for wide character arrays
- [“vwprintf” on page 560](#) behaves like `vprintf` for wide character arrays
- [“vswprintf” on page 559](#) behaves like `fgetc vsprintf` for wide character arrays
- [“vwscanf” on page 557](#) behaves like `vscanf` for wide character arrays

Time facilities

- [“wcsftime” on page 566](#) behaves like `csftime` for wide character arrays
- [“wctime” on page 582](#) behaves like `ctime` for wide character arrays

String facilities

- [“watof” on page 561](#) behaves like `atof` for wide characters array
- [“wcscat” on page 562](#) behaves like `strcat` for wide character arrays
- [“wcschr” on page 563](#) behaves like `strchr` for wide character arrays
- [“wcscmp” on page 563](#) behaves like `strcmp` for wide character arrays
- [“wcscspn” on page 564](#) behaves like `strspn` for wide character arrays
- [“wcscoll” on page 564](#) behaves like `strcoll` for wide character arrays

- [“wcscpy” on page 565](#) behaves like `strcpy` for wide character arrays
- [“wcslen” on page 566](#) behaves like `strlen` for wide character arrays
- [“wcsncat” on page 567](#) behaves like `strncat` for wide character arrays
- [“wcsncmp” on page 568](#) behaves like `strcmp` for wide character arrays
- [“wcsncpy” on page 568](#) behaves like `strncpy` for wide character arrays
- [“wcspbrk” on page 569](#) behaves like `strbrk` for wide character arrays
- [“wcsrchr” on page 569](#) behaves like `strrchr` for wide character arrays
- [“wcsspn” on page 571](#) behaves like `strspn` for wide character arrays
- [“wcsstr” on page 572](#) behaves like `strstr` for wide character arrays
- [“wcstod” on page 572](#) behaves like `strtod` for wide character arrays
- [“wcstof” on page 573](#) behaves like `strtod` for wide character arrays
- [“wcstok” on page 574](#) behaves like `strtok` for wide character arrays
- [“wcstol” on page 575](#) behaves like `strtol` for wide character arrays
- [“wcstold” on page 577](#) behaves like `strtold` for wide character arrays
- [“wcstoll” on page 578](#) behaves like `strtoll` for wide character arrays
- [“wcstoul” on page 579](#) behaves like `strtoul` for wide character arrays
- [“wcstoull” on page 580](#) behaves like `strtoull` for wide character arrays
- [“wcsxfrm” on page 582](#) behaves like `strxfrm` for wide character arrays

- [“wmemcmpr” on page 583](#) behaves like `memchr` for wide character arrays
- [“wmemcpy” on page 585](#) behaves like `memcpy` for wide character arrays
- [“wmemcmp” on page 584](#) behaves like `memcmp` for wide character arrays
- [“wmemmove” on page 585](#) behaves like `memmove` for wide character arrays
- [“wmemset” on page 586](#) behaves like `memset` for wide character arrays

Multibyte character functions

- [“mbrlen” on page 549](#) behaves like `strlen` for multibyte character arrays
- [“mbrtowc” on page 550](#) converts multibyte characters to wide character arrays
- [“mbsrtowcs” on page 551](#) converts multibyte strings to wide character strings
- [“wcrtomb” on page 561](#) converts wide characters to multibyte character arrays.
- [“wcsrtombs” on page 570](#) converts wide character strings to multibyte strings.

Conversion functions

- [“btowc” on page 543](#) converts byte characters to wide character arrays
- [“wctob” on page 583](#) 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.

Table 43.1 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	vscanf	vprintf

Table 43.2 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	wcscpy	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.

Table 43.3 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

Defines	Definitions
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 character representation and WEOF

btowc

The function `btowc()` converts a single byte character to a wide character.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wchar.h> wint_t btowc(int c);	
Parameters	Parameters for this function are:	
	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.	
See Also	“wctob” on page 583	

fgetwc

Gets a wide character from a file stream.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wchar.h> wchar_t fgetwc(FILE * file);	
Parameters	Parameters for this function are:	

	file	FILE *	The input stream to retrieve from
Remarks	Performs the same task as <code>fgetc</code> for wide character		
	On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.		
Return	Returns the character or <code>WEOF</code> for an error		
See Also	“Wide Character and Byte Character Stream Orientation” on page 540 “fgetc” on page 311		

fgetws

The function `fgetws()` reads a wide character string from a file stream.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	<pre>#include <wchar.h> wchar_t *fgetws(wchar_t * s, int n, FILE * file);</pre>		
Parameters	Parameters for this function are:		
	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 <code>fgets()</code> for wide characters.		
	On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.		
Return	Returns a pointer to <code>s</code> if successful or <code>NULL</code> for a failure.		
See Also	“Wide Character and Byte Character Stream Orientation” on page 540 “fgets” on page 315		

fputwc

Inserts a single wide character into a file stream.

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

Prototype

```
#include <wchar.h>
wchar_t fputwc(wchar_t c, FILE * file);
```

Parameters Parameters for this function are:

c	wchar_t	The 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.

Return Returns the character written if it is successful, and returns WEOF if it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 540](#)

[“fputc” on page 327](#)

fputws

Inserts a wide character array into a file stream

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

Prototype

```
#include <wchar.h>
int fputws(const wchar_t * s, FILE * file);
```

Parameters Parameters for this function are:

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.

Return Returns a zero if successful, and returns a nonzero value when it fails.

See Also [“Wide Character and Byte Character Stream Orientation” on page 540](#)

[“fputs” on page 328](#)

fwprintf

Formatted file insertion

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

Prototype

```
#include <wchar.h>
int fwprintf(FILE * file,
             const wchar_t * format, ...);
```

Parameters Parameters for this function are:

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 586](#) function for details of the format string.

On embedded/RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return Returns the number of arguments written or a negative number if an error occurs

See Also ["Wide Character and Byte Character Stream Orientation" on page 540](#)
["wprintf" on page 586](#)

fwscanf

Reads formatted text from a stream.

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

Prototype

```
#include <wchar.h>
int fwscanf(FILE * file,
            const wchar_t * format, ...);
```

Parameters Parameters for this function are:

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 590](#) function for details of the format string.

On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return 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.

See Also [“Wide Character and Byte Character Stream Orientation” on page 540](#)
[“wscanf” on page 590](#)

getwc

Reads the next character from a wide character stream.

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

Prototype `#include <wchar.h>`
`wchar_t getwc(FILE * file);`

Parameters Parameters for this function are:

file FILE * The file stream

Remarks Performs the same task as `getc` for a wide character type.

Return Returns the next character from the stream or returns WEOF if the end-of-file has been reached or a read error has occurred.

See Also [“Wide Character and Byte Character Stream Orientation” on page 540](#)
[“getc” on page 347](#)

getwchar

Returns a wide character type from the standard input.

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

Prototype	<pre>#include <wchar.h> wchar_t getwchar(void);</pre>
Parameters	Has no parameters.
Remarks	Performs the same task as <code>getchar</code> for a wide character type.
Return	Returns the value of the next character from <code>stdin</code> as an <code>int</code> if it is successful. <code>getwchar()</code> returns <code>WEOF</code> if it reaches an end-of-file or an error occurs.
See Also	“Wide Character and Byte Character Stream Orientation” on page 540 “getwchar” on page 548

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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.									
Prototype	<pre>#include <stdlib.h> int mbrlen(const char *s, size_t n, mbstate_t * ps);</pre>									
Parameters	Parameters for this function are: <table><tr><td>s</td><td>const char **</td><td>The multibyte array to measure</td></tr><tr><td>n</td><td>size_t</td><td>The Maximum size</td></tr><tr><td>ps</td><td>mbstate_t **</td><td>The current state of translation between multibyte and wide-character, ignored if the encoding scheme is non-modal.</td></tr></table>	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.
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 <code>mbrlen()</code> function returns the length of the multibyte character pointed to by <code>s</code> . It examines a maximum of <code>n</code> characters.									

The Metrowerks C implementation supports the "C" locale with UTF-8 encoding only and returns the value of `mbrtowc(NULL, s, n, pc)`.

Return `mbrlen()` returns the value of `mbrtowc(NULL, s, n, pc)`.

See Also ["wcslen" on page 566](#)

["strlen" on page 459](#)

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.

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

Prototype

```
#include <stdlib.h>
int mbrtowc(wchar_t *pwc,
            const char *s, size_t n, mbstate_t * ps);
```

Parameters Parameters for this function are:

pwc	<code>wchar_t *</code>	The wide character destination
s	<code>const char *</code>	The string to convert
n	<code>size_t</code>	The maximum wide characters to convert
ps	<code>mbstate_t *</code>	The current state of translation between multibyte and wide-character, ignored if the encoding scheme is non-modal.

Remarks If `s` 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.

Return `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.

See Also

[“mbsrtowcs” on page 551](#)

[“wcrtomb” on page 561](#)

[“wcsrtombs” on page 570](#)

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.

Compatibility

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

Prototype

```
#include <stdlib.h>
size_t mbsrtowcs(wchar_t *pwcs,
                  const char **s, size_t n, mbstate_t * ps);
```

Parameters

Parameters for this function are:

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 <code>mbsrtowcs()</code> converts a sequence of multibyte characters encoded according to the scheme specified in the <code>LC_CTYPE</code> component of the current locale from the character array indirectly pointed to by <code>s</code> and, if <code>pwcs</code> is not a null pointer, stores not more than <code>n</code> of the corresponding Unicode characters into the wide-character array pointed to by <code>pwcs</code> . 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 <code>s</code> , otherwise a pointer to the address just beyond the last multibyte character converted, if any.
Return	If an invalidly encoded character is encountered, <code>mbsrtowcs()</code> stores the value of <code>EILSEQ</code> in <code>errno</code> and returns the value <code>(size_t) (-1)</code> . Otherwise <code>mbsrtowcs()</code> returns the number of multibyte characters successfully converted, not including any terminating null wide character.
See Also	“wcsrtombs” on page 570 “mbrtowc” on page 550

putwc

Write a wide character type to a stream.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <wchar.h> wchar_t putwc(wchar_t c, FILE * file);</pre>
Parameters	Parameters for this function are:

	c	wchar_t	The wide character to output
	file	FILE	The output stream
Remarks	Performs the same task as <code>putc</code> for a wide character type.		
Return	Returns the character written when successful and returns <code>WEOF</code> when it fails		
See Also	“Wide Character and Byte Character Stream Orientation” on page 540 “putc” on page 361		

putwchar

Writes a wide character to standard output.

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

Prototype `#include <wchar.h>`
`wchar_t putwchar(wchar_t c);`

Parameters Parameters for this function are:

c wchar_t The wide character to write.

Remarks Performs the same task as `putchar` for a wide character type.

Return Returns `c` if it is successful and returns `WEOF` if it fails

See Also [“Wide Character and Byte Character Stream Orientation” on page 540](#)
[“putchar” on page 363](#)

swprintf

Formats text in a wide character type string.

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

Prototype `#include <wchar.h>`

```
int swprintf(wchar_t * s, size_t N,
             const wchar_t * format, ...);
```

Parameters Parameters for this function are:

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 586](#) function for details of the format string.

Return Returns the number of characters assigned to `s`, not including the null character, or a negative number if `n` or more characters were requested to be written.

See Also ["Wide Character and Byte Character Stream Orientation" on page 540](#)
["fwprintf" on page 546](#)
["sprintf" on page 379](#)

swscanf

Reads a formatted wide character string.

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

Prototype

```
#include <wchar.h>
int swscanf(const wchar_t * s,
            const wchar_t * format, ...);
```

Parameters	Parameters for this function are:		
	s	wchar_t*	The string being read
	format	wchar_t*	The format string
		Variable arguments
Remarks	Performs the same task as <code>sscanf</code> for a wide character type.		
Return	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 590 function for details of the format string.		
See Also	"Wide Character and Byte Character Stream Orientation" on page 540 "sscanf" on page 381		

vfwscanf

Read formatted text from a wide-character stream.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	#include <wchar.h> int vfwscanf(FILE * file, const wchar_t * format_str, va_list arg);		
Parameters	Parameters for this function are:		
	file	FILE *	The stream being read
	format_str	const wchar_t*	The format string
		Variable arguments
Remarks	Performs the same task as <code>fscanf</code> for a wide character type.		
	Refer to the " wscanf " on page 590 function for details of the format string.		

NOTE On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return `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.

See Also [“Wide Character and Byte Character Stream Orientation” on page 540](#)
[“fscanf” on page 333](#)

vswscanf

Reads formatted text from a wide-character string.

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

Prototype

```
#include <wchar.h>
int __vswscanf(const wchar_t * s,
                const wchar_t * format, va_list arg);
```

Parameters Parameters for this function are:

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 590](#) function for details of the format string.

NOTE On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return `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`.

See Also [“Wide Character and Byte Character Stream Orientation” on page 540](#)
[“sscanf” on page 381](#)

vwscanf

Reads formatted text from wide-character oriented stdin.

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

Prototype

```
#include <wchar.h>
int vwscanf(const wchar_t * format, va_list arg);
```

Parameters Parameters for this function are:

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 590](#) function for details of the format string.

Return 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`.

See Also

[“Wide Character and Byte Character Stream Orientation” on page 540](#)

[“vfscanf” on page 555](#)

[“scanf” on page 369](#)

vwprintf

Write a formatted text to a file stream.

Compatibility

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

Prototype

```
#include <wchar.h>
int vfwprintf(FILE * file,
               const wchar_t * format_str, va_list arg);
```

Parameters

Parameters for this function are:

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 `fprintf()` function. Instead of the variable list of arguments that can be passed to `fprintf()`, `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 586](#) function for details of the format string.

NOTE

On embedded/ RTOS systems this function only is implemented for `stdin`, `stdout` and `stderr` files.

Return Returns the number of wide characters written or a negative number if it failed.

See Also [“Wide Character and Byte Character Stream Orientation” on page 540](#)
[“vfprintf” on page 386](#)

vswprintf

Write a formatted output to a wide character string.

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

Prototype

```
#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);
```

Parameters Parameters for this function are:

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 586](#) function for details of the format string.

NOTE The `vswprintf` function does not invoke the `va_end` macro.

Return	Returns the number of characters written not counting the terminating null wide character. Otherwise a negative value if a failure occurs.
See Also	“Wide Character and Byte Character Stream Orientation” on page 540 “swprintf” on page 553 “vsprintf” on page 394

vwprintf

Write a formatted text to a wide-character oriented stdout

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.						
Prototype	#include <wchar.h> int vwprintf(const wchar_t * format, va_list arg);						
Parameters	Parameters for this function are: <table><tr><td>format</td><td>wchar_t*</td><td>The format string</td></tr><tr><td>....</td><td></td><td>Variable arguments</td></tr></table>	format	wchar_t*	The format string		Variable arguments
format	wchar_t*	The format string					
....		Variable arguments					
Remarks	The <code>vwprintf()</code> function works identically to the <code>wprintf()</code> function. Instead of the variable list of arguments that can be passed to <code>wprintf()</code> , <code>vwprintf()</code> accepts its arguments in the array <code>arg</code> of type <code>va_list</code> , which must have been initialized by the <code>va_start()</code> macro (and possibly subsequent <code>va_arg</code> calls) from the <code>stdarg.h</code> header file. The <code>vwprintf()</code> function does not invoke the <code>va_end</code> macro. Refer to the “wprintf” on page 586 function for details of the format string.						
Return	Returns the number of characters written or a negative value if it failed.						
See Also	“Wide Character and Byte Character Stream Orientation” on page 540 “vprintf” on page 391						

watof

Convert a wide character string to a double type

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

Prototype

```
#include <wchar.h>
double watof(wchar_t * str);
```

Parameters Parameters for this function are:

str	wchar_t	The wide character string to be converted
-----	---------	---

Remarks Performs the same task as `atof` for a wide character type.

Return Returns the converted value or, if no conversion could be performed, zero.

See Also [“atof” on page 405](#)

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.

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

Prototype

```
#include <stdlib.h>
int wcrtomb
    (char *s, wchar_t wchar, mbstate_t * ps);
```

Parameters Parameters for this function are:

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.		
Return	wcrtomb () returns the number of bytes stored in the string s .		
See Also	“mbrtowc” on page 550 “wcsrtombs” on page 570		

wcscat

Wide character string concatenation

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	#include <wchar.h> wchar_t * wcscat(wchar_t * dst, const wchar_t * src);
Parameters	Parameters for this function are:
	dst wchar_t * The destination string
	src wchar_t * The source string
Remarks	Performs the same task as strcat for a wide character type.
Return	Returns a pointer to the destination string
See Also	“strcat” on page 451

wcschr

Search for an occurrence of a wide character.

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

Prototype

```
#include <wchar.h>
wchar_t * wcschr(const wchar_t * str,
                 const wchar_t chr);
```

Parameters Parameters for this function are:

str	wchar_t *	The string to be searched
chr	wchar_t	The character to search for

Remarks Performs the same task as `strchr` for a wide character type.

Return Returns a pointer to the successfully located character. If it fails, `wcschr()` returns a null pointer (`NULL`).

See Also [“strchr” on page 452](#)

wcscmp

Compare two wide character arrays.

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

Prototype

```
#include <wchar.h>
int cscmp(const wchar_t * str1,
          const wchar_t * str2);
```

Parameters Parameters for this function are:

str1	wchar_t *	Comparison string
str2	wchar_t *	Comparison string

Remarks Performs the same task as `strcmp` for a wide character type.

Return 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`.

See Also [“strcmp” on page 453](#)
 [“wmemcmp” on page 584](#)

wcscoll

Compare two wide character arrays according to the collating sequence defined in the LC_COLLATE component of the current locale.

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

Prototype

```
#include <wchar.h>
int wcscoll(const wchar_t *str1,
            const wchar_t * str2);
```

Parameters Parameters for this function are:

str1	wchar_t *	First comparison string
str2	wchar_t *	Second comparison string

Remarks Performs the same task as `strcoll` for a wide character type.

Return 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`.

See Also [“strcoll” on page 454](#)
 [“wcscmp” on page 563](#)
 [“wmemcmp” on page 584](#)

wcscspn

Find the first character in one wide character string that is also in another.

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

Prototype

```
#include <wchar.h>
size_t wcscspn(const wchar_t * str,
                const wchar_t * set);
```

Parameters	Parameters for this function are:		
	str	wchar_t *	The string to be searched
	set	wchar_t *	The set of characters to be searched for
Remarks	The <code>wcsncpy()</code> function finds the first character in the null terminated wide-character string <code>s1</code> that is also in the null terminated wide character string <code>s2</code> . For this purpose, the null terminators are considered part of the strings. The function starts examining characters at the beginning of <code>s1</code> and continues searching until a character in <code>s1</code> matches a character in <code>s2</code> .		
Return	<code>wcsncpy()</code> returns the index of the first character in <code>s1</code> that matches a character in <code>s2</code> .		
See Also	"strncpy" on page 457		

wcsncpy

Copy one wide character array to another.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	<pre>#include <wchar.h> wchar_t * (wcsncpy) (wchar_t * dst, const wchar_t * src);</pre>		
Parameters	Parameters for this function are:		
	dst	wchar_t *	The destination string
	src	wchar_t *	The source being copied
Remarks	The <code>wcsncpy()</code> function copies the character array pointed to by <code>src</code> to the character array pointed to <code>dst</code> . The <code>src</code> argument must point to a null terminated wide character array. The resulting character array at <code>dest</code> is null terminated as well. If the arrays pointed to by <code>dest</code> and <code>source</code> overlap, the operation of <code>strcpy()</code> is undefined.		
Return	Returns a pointer to the destination string.		

See Also [“strcpy” on page 456](#)

wcsftime

Formats a wide character string for time.

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

Prototype

```
#include <wchar.h>
size_t wcsftime(wchar_t * str,
                 size_t max_size,
                 const wchar_t * format_str,
                 const struct tm * timeptr);
```

Parameters Parameters for this function are:

str	wchar_t *	The destination string
max_size	size_t	Maximum string size
format_str	const wchar_t *	The format string
timeptr	const struct tm *	The time structure containing the calendar time

Remarks Performs the same task as `strftime` for a wide character type.

Return 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

See Also [“strftime” on page 485](#)

wcslen

Compute the length of a wide character array.

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

Prototype

```
#include <wchar.h>
size_t (wcslen)(const wchar_t * str);
```

Parameters	Parameters for this function are:	
	str	wchar_t * The string to compute
Remarks	The <code>wcslen()</code> function computes the number of characters in a null terminated wide-character array pointed to by <code>str</code> . The null character (<code>L'\0'</code>) is not added to the character count.	
Return	Returns the number of characters in a wide-character array not including the terminating null character.	
See Also	"strlen" on page 459	

wcsncat

Append a specified number of characters to a wide character array.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	<pre>#include <wchar.h> wchar_t * wcsncat(wchar_t * dst, const wchar_t * src, size_t n);</pre>	
Parameters	Parameters for this function are:	
	dst	wchar_t * The destination string
	src	wchar_t * The string to be appended
	n	size_t The number of characters to copy
Remarks	<p>The <code>wcsncat()</code> function appends a maximum of <code>n</code> characters from the character array pointed to by <code>source</code> to the character array pointed to by <code>dest</code>. The <code>dest</code> argument must point to a null terminated character array. The <code>src</code> argument does not necessarily have to point to a null terminated character array.</p> <p>If a null character is reached in <code>src</code> before <code>n</code> characters have been appended, <code>wcsncat()</code> stops.</p> <p>When done, <code>wcsncat()</code> terminates <code>dest</code> with a null character (<code>L'\0'</code>).</p>	
Return	Returns a pointer to the destination string.	

See Also [“strncat” on page 460](#)

wcsncmp

Compare not more than a specified number of wide characters.

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

Prototype

```
#include <wchar.h>
int csncmp(const wchar_t * str1,
           const wchar_t * str2, size_t n);
```

Parameters Parameters for this function are:

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.

Return 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`.

See Also [“strcmp” on page 461](#)

wcsncpy

Copy a specified number of wide characters.

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

Prototype

```
#include <wchar.h>
wchar_t * wcsncpy(wchar_t * dst,
                  const wchar_t * src, size_t n);
```

Parameters Parameters for this function are:

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 <code>strncpy</code> for a wide character type.	
Return	Returns a pointer to the destination string.	
See Also	“<code>strncpy</code>” on page 462 “<code>wcsncpy</code>” on page 565	

wcspbrk

Look for the first occurrence of an element of an array of wide characters in another.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.						
Prototype	<pre>#include <wchar.h> wchar_t * wcspbrk(const wchar_t * str, const wchar_t * set);</pre>						
Parameters	Parameters for this function are:						
	<table><tr><td>str</td><td>wchar_t*</td><td>The string being searched</td></tr><tr><td>set</td><td>wchar_t *</td><td>The search set</td></tr></table>	str	wchar_t*	The string being searched	set	wchar_t *	The search set
str	wchar_t*	The string being searched					
set	wchar_t *	The search set					
Remarks	Performs the same task as <code>strupbrk</code> for a wide character type.						
Return	Returns a pointer to the first character in <code>str</code> that matches any character in <code>set</code> , and returns a null pointer (NULL) if no match was found.						
See Also	“<code>strupbrk</code>” on page 464						

wcsrchr

Search a wide character string for the last occurrence of a specified wide character.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wchar.h> wchar_t * wcsrchr(const wchar_t * str, wchar_t chr);	
Parameters	Parameters for this function are:	
	str	const wchar_t * The string being searched
	chr	wchar_t The character to search for
Remarks	Performs the same task as <code>strrchr</code> for a wide character type.	
Return	Returns a pointer to the character found or returns a null pointer (NULL) if it fails.	
See Also	"strrchr" on page 465	

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.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wchar.h> size_t wcsrtombs (const char **src, size_t n, mbstate_t * ps);	
Parameters	Parameters for this function are:	
	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	<p>The MSL implementation of the <code>wcsrtombs()</code> function converts a character array containing <code>wchar_t</code> type Unicode characters indirectly pointed to by <code>src</code> to a character array containing UTF-8 multibyte characters. If <code>dst</code> is not a null pointer, these multibyte characters are stored in the array pointed to by <code>dst</code>. Conversion continues until either a terminating null wide-character is encountered or (if <code>dst</code> 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 <code>n</code>.</p> <p>If <code>dst</code> is not a null pointer, the <code>wchar_t *</code> object pointed to by <code>src</code> 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.</p>		
Return	<p><code>wcsrtombs()</code> returns the number of bytes modified in the character array pointed to by <code>s</code>, not including a terminating null character, if any.</p>		
See Also	<p>“wcrtomb” on page 561 “mbsrtowcs” on page 551</p>		

WCSSPN

Count the number of wide characters in one wide character array that are in another.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <wchar.h> size_t wcsspn(const wchar_t * str, const wchar_t * set);</pre>
Parameters	Parameters for this function are:

	str wchar_t * The searched string
	set const wchar_t * The search set
Remarks	Performs the same task as <code>strspn</code> for a wide character type.
Return	Returns the number of characters in the initial segment of <code>str</code> that contains only characters that are elements of <code>set</code> .
See Also	"strspn" on page 465

wcsstr

Search for a wide character array within another.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <wchar.h> wchar_t * wcsstr(const wchar_t * str, const wchar_t * pat);</pre>
Parameters	Parameters for this function are:
	str const wchar_t * The string to search
	pat const wchar_t * The string being searched for
Remarks	Performs the same task as <code>strstr</code> for a wide character type.
Return	Returns a pointer to the first occurrence of <code>s2</code> in <code>s1</code> and returns a null pointer (NULL) if <code>s2</code> cannot be found.
See Also	"strstr" on page 467 "wcschr" on page 563

wcstod

Converts a wide character array to double values.

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

Prototype	#include <wchar.h> double wcstod(wchar_t * str, char ** end);		
Parameters	Parameters for this function are: str wchar_t * The string being converted end char ** If not null, a pointer to the first position not convertible.		
Remarks	The <code>wcstod()</code> function converts a character array, pointed to by <code>nptr</code> , to a floating point value of type <code>float</code> . The character array can be in either decimal or hexadecimal floating point constant notation (e.g. <code>103.578</code> , <code>1.03578e+02</code> , or <code>0x1.9efef9p+6</code>). If the <code>endptr</code> argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by <code>nptr</code> . This position marks the first character that is not convertible to a value of type <code>double</code> .		
	In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.		
	This function skips leading white space.		
Return	Returns a floating point value of type <code>double</code> . If <code>str</code> cannot be converted to an expressible <code>double</code> value, <code>wcstod()</code> returns <code>HUGE_VAL</code> , defined in <code>math.h</code> , and sets <code>errno</code> to <code>ERANGE</code>		
See Also	"wcstof" on page 573 "strtod" on page 428 "wcstold" on page 577 "errno" on page 83		
	wcstof		
	Wide character array conversion to floating point value of type <code>float</code> .		
Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.		
Prototype	#include <wchar.h>		

```
float wcstof(const wchar_t * restrict nptr,  
            wchar_t ** restrict endptr);
```

Parameters Parameters for this facility are:

nptr	const char *	A Null terminated wide character array to convert
endptr	char **	A pointer to a position in nptr that is follows the converted part.

Remarks The `wcstof()` 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.

Return `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`.

See Also ["wcstod" on page 572](#)
["wcstold" on page 577](#)
["strtod" on page 430](#)
["errno" on page 83](#)

wcstok

Extract tokens within a wide character array.

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

Prototype `#include <wchar.h>`

```
wchar_t * wcstok(wchar_t * str,  
                  const wchar_t * set, wchar_t ** ptr);
```

Parameters	Parameters for this function are:		
	str	wchar_t *	The string to be modified
	set	wchar_t *	The list of character to find
	ptr	wchar_t *	Continuation information
Remarks	Performs the same task as <code>strtok</code> 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. <code>wcstok()</code> returns a pointer to the first token in <code>str</code> or returns a null pointer if no token can be found.		
	Subsequent calls to <code>wcstok()</code> with a NULL <code>str</code> argument causes <code>wcstok()</code> to return a pointer to the next token or return a null pointer (NULL) when no more tokens exist. When called with a NULL <code>str</code> argument, the value of the <code>ptr</code> argument must have been set by a previous call to <code>wcstok()</code> .		
Return	The <code>wcstok</code> function returns a pointer to the first wide character of a token, or a null pointer if there is no token.		
See Also	"strtok" on page 468 "errno" on page 83		

wcstol

Wide character array conversion to floating point value of type long int.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.
Prototype	<pre>#include <wchar.h> long int wcstol(const wchar_t * restrict nptr, wchar_t ** restrict endptr, int base);</pre>
Parameters	Parameters for this facility are:

nptr	const char *	A Null terminated wide character array to convert
endptr	char **	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 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 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 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.

Return `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`

See Also ["strtol" on page 431](#)
["errno" on page 83](#)
["wcstoll" on page 578](#)
["wcstoul" on page 579](#)

wcstold

Character array conversion to an floating point value of type long double.

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

Prototype

```
#include <wchar.h>
long double wcstold(const wchar_t * restrict nptr,
                     wchar_t ** restrict endptr);
```

Parameters Parameters for this facility are:

nptr	const char *	A Null terminated wide character array to convert
endptr	char **	A pointer to a position in nptry that is not convertible.

Remarks The `wcstold()` 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`. A plus or minus sign (+ or -) prefixing the number string is optional. 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 double value.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

Return 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`.

See Also ["wcstod" on page 572](#)
["wcstof" on page 573](#)
["strtold" on page 434](#)

wcstoll

Wide character array conversion to integer value of type long long int.

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

Prototype

```
#include <wchar.h>
long long int wcstoll(const wchar_t * restrict
nptr,
    wchar_t ** restrict endptr, int base);
```

Parameters Parameters for this facility are:

nptr	const char *	A Null terminated wide character 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 `wcstoll()` 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. 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 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 long int value.

In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.

This function skips leading white space.

Return `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`

See Also [“`strtoll`” on page 435](#)
[“`wcstol`” on page 575](#)
[“`wcstoul`” on page 579](#)
[“`errno`” on page 83](#)

wcstoul

Wide character array conversion to integer value of type `unsigned long int`.

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

Prototype

```
#include <wchar.h>
unsigned long int wcstoul(
    const wchar_t * restrict nptr,
    wchar_t ** restrict endptr, int base);
```

Parameters Parameters for this facility are:

<code>nptr</code>	<code>const char *</code>	A Null terminated wide character array to convert
<code>endptr</code>	<code>char **</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 `wcstoul()` function converts a 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 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.

Return `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`

See Also

["wcstol" on page 575](#)
["wcstoll" on page 578](#)
["wcstoull" on page 580](#)
["strtoul" on page 436](#)
["errno" on page 83](#)

wcstoull

Wide character array conversion to integer value of type `unsigned long long int`.

Compatibility

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

Prototype

```
#include <wchar.h>
unsigned long long int wcstoull(
    const wchar_t * restrict nptr,
    wchar_t ** restrict endptr, int base);
```

Parameters	Parameters for this facility are:	
	nptr const char *	A Null terminated wide character 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	<p>The <code>wcstoull()</code> function converts a character array, pointed to by <code>nptr</code>, expected to represent an integer expressed in radix base to an integer value of type <code>unsigned long long int</code>. A plus or minus sign prefix is ignored.</p> <p>The base argument in <code>wcstoull()</code> 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 <code>wcstoull()</code> 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.</p> <p>If the <code>endptr</code> argument is not a <code>null</code> pointer, it is assigned a pointer to a position within the character array pointed to by <code>nptr</code>. This position marks the first character that is not convertible to a <code>long int</code> value.</p> <p>In other than the "C" locale, additional locale-specific subject sequence forms may be accepted.</p> <p>This function skips leading white space.</p>	
Return	<p><code>wcstoull()</code> returns an <code>unsigned integer</code> value of type <code>unsigned long long int</code>. If the converted value is greater than <code>ULLONG_MAX</code>, <code>wcstoull()</code> returns <code>ULLONG_MAX</code> and sets <code>errno</code> to <code>ERANGE</code>. The <code>ULLONG_MAX</code> macro is defined in <code>limits.h</code></p>	
See Also	<p>"wcstol" on page 575 "wcstoll" on page 578 "wcstoul" on page 579 "strtoull" on page 438</p>	

[“errno” on page 83](#)

wcsxfrm

Transform a wide character array as specified in the `LC_COLL` component of the current locale.

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

Prototype

```
#include <wchar.h>
size_t wcsxfrm(wchar_t * str1,
               const wchar_t * str2, size_t n);
```

Parameters Parameters for this function are:

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.

Return 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.

See Also [“strxfrm” on page 470](#)

wctime

Convert a `time_t` type to a wide character array

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

Prototype

```
#include <wchar.h>
wchar_t * wctime(const time_t * timer);
```

Parameters Parameters for this function are:

timer	const time_t *	The Calendar Time
-------	----------------	-------------------

Remarks Performs the same task as `ctime` for a wide character type.

Return Returns a pointer to wide character array containing the converted time_t type

See Also [“ctime” on page 481](#)

wctob

The function wctob() converts a wide character to a byte character.

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

Prototype

```
#include <wchar.h>
    int wctob(wint_t wc);
```

Parameters Parameters for this function are:

int wc * 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.

See Also [“wcrtomb” on page 561](#)

wmemchr

Search for an occurrence of a specific wide character.

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

Prototype

```
#include <wchar.h>
void * wmemchr(const void * src,
    int val, size_t n);
```

Parameters Parameters for this function are:

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 <code>memchr()</code> for a wide character type.
Return	Returns a pointer to the found wide character, or a null pointer (<code>NULL</code>) if <code>val</code> cannot be found.
See Also	“memchr” on page 446 “wcschr” on page 563

wmemcmp

Compare two blocks of memory, treated as wide characters.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.									
Prototype	<pre>#include <wchar.h> int wmemcmp(const void * src1, const void * src2, size_t n);</pre>									
Parameters	Parameters for this function are: <table><tr><td>src1</td><td>const void *</td><td>First memory block to compare</td></tr><tr><td>src2</td><td>const void *</td><td>Second memory block to compare</td></tr><tr><td>n</td><td>size_t</td><td>Maximum number of wide characters to compare</td></tr></table>	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
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 <code>memcmp()</code> for a wide character type.									
Return	The function <code>wmemcmp</code> returns a zero if all <code>n</code> characters pointed to by <code>src1</code> and <code>src2</code> are equal. The function <code>wmemcmp</code> returns a negative value if the first non-matching character pointed to by <code>src1</code> is less than the character pointed to by <code>src2</code> . The function <code>wmemcmp</code> returns a positive value if the first non-matching character pointed to by <code>src1</code> is greater than the character pointed to by <code>src2</code> .									
See Also	“memcmp” on page 448 “wcscmp” on page 563									

wmemcpy

Copy a contiguous memory block.

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

Prototype

```
#include <wchar.h>
void * (wmemcpy) (void * dst,
                  const void * src, size_t n);
```

Parameters Parameters for this function are:

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.

Return Returns a pointer to the destination string.

See Also [“memcpy” on page 449](#)

wmemmove

Copy an overlapping contiguous memory block.

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

Prototype

```
#include <wchar.h>
void * (wmemmove) (void * dst,
                   const void * src,
                   size_t n);
```

Parameters Parameters for this function are:

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.

Return Returns a pointer to the destination string.

See Also [“memmove” on page 450](#)

[“wcscpy” on page 565](#)

wmemset

Clear the contents of a block of memory.

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

Prototype

```
#include <wchar.h>
void * wmemset(void * dst, int val, size_t n);
```

Parameters Parameters for this function are:

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.

Return Returns a pointer to the destination string

See Also [“memset” on page 451](#)

wprintf

Send formatted wide character text to a standard output.

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

Prototype

```
#include <wchar.h>
int wprintf(const wchar_t * format, ...);
```

Parameters Parameters for this function are:

format	wchar_t*	The format string
....		Variable arguments

Remarks	Performs the same task as <code>printf()</code> for a wide character type.
Return	Returns the number of arguments written or a negative number if an error occurs.

Table 43.4 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.
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	AltiVec: A vector short, vector unsigned short, vector bool short or vector pixel when followed by c, d, i, o, u, x or X
hv	

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.
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 <code>printf()</code> so far. Its corresponding argument must be a pointer to an <code>int</code> .
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 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 (<code>float</code> or <code>double</code>) 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.
s	If the s format specifier is preceded by an l length modifier, the argument shall be a pointer to the initial element of an array of wchar_t type that is null terminated If there is no l 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 x 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.
----	---

See Also

[“Wide Character and Byte Character Stream Orientation” on page 540](#)

[“printf” on page 353](#)

[“fwprintf” on page 546](#)

wscanf

Reads a wide character formatted text from standard input

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wchar.h> int wscanf(const wchar_t * format, ...);	
Parameters	Parameters for this function are: format wchar_t* The format string Variable arguments	
Remarks	Performs the same task as <code>scanf()</code> for a wide character type.	
Return	Returns the number of items successfully read and returns <code>WEOF</code> if a conversion type does not match its argument or and end-of-file is reached.	

Table 43.5 Length Modifiers And Conversion Specifiers For Formatted 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	AltiVec: vector short, vector unsigned short, vector bool short or vector pixel
hv	when followed by c, d, i, o, u, x or X
vl	AltiVec: vector long, vector unsigned long
lv	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.aaae \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 590](#), for a full description.

See Also

- [“Wide Character and Byte Character Stream Orientation” on page 540](#)
- [“scanf” on page 369](#)
- [“fwscanf” on page 547](#)

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 113, for a full description.
- For the function `wcsicmp` see “[wcsicmp](#)” on page 113, for a full description.
- For the function `wcslwr` see “[wcslwr](#)” on page 114, for a full description.
- For the function `wcsncmp` see “[wcsncmp](#)” on page 116, for a full description.
- For the function `wcsnset` see “[wcsnset](#)” on page 117, for a full description.
- For the function `wcsrev` see “[wcsrev](#)” on page 118, for a full description.
- For the function `wcsset` see “[wcsset](#)” on page 118, for a full description.
- For the function `wcsspnp` see “[wcsspnp](#)” on page 119, for a full description.
- For the function `wcsupr` see “[wcsupr](#)” on page 119, for a full description.
- For the function `wtoi` see “[wtoi](#)” on page 120, 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 596](#) defines two types used for wide character values
- [“iswalnum” on page 596](#) tests for alpha-numeric wide characters
- [“iswalpha” on page 597](#) tests for alphabetical wide characters
- [“iswblank” on page 597](#) tests for a blank space or space holder
- [“iswcntrl” on page 598](#) tests for control wide characters
- [“iswdigit” on page 598](#) tests for digital wide characters
- [“iswgraph” on page 599](#) tests for graphical wide characters
- [“iswlower” on page 599](#) tests for lower wide characters
- [“iswprint” on page 599](#) tests for printable wide characters
- [“iswpunct” on page 600](#) tests for punctuation wide characters
- [“iswspace” on page 600](#) tests for whitespace wide characters
- [“iswupper” on page 601](#) tests for uppercase wide characters
- [“iswxdigit” on page 601](#) tests for a hexadecimal wide character type
- [“towlower” on page 602](#) converts wide characters to lower case
- [“towupper” on page 603](#) converts wide characters to upper case

Mapping facilities

- [“towctrans” on page 602](#) a case and wide character mapping function
- [“wctrans” on page 603](#) a wide character mapping function

Types

The header `wctype.h` contains two types described in the table [“wctype.h types” on page 596](#) used for wide character manipulations.

Table 44.1 `wctype.h` types

<code>wctrans_t</code>	A scalar type which can represent locale specific character mappings
<code>wctype_t</code>	A scalar type that can represent locale specific character classifications

iswalnum

Tests for alpha-numeric wide characters.

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

Prototype `#include <wctype.h>`
`int iswalnum (wchar_t wc);`

Parameters The parameter for this function is:

<code>wc</code>	<code>wchar_t</code>	The wide character to test
-----------------	----------------------	----------------------------

Remarks Provides the same functionality as `isalnum` for wide character type.

Return In the “C” locale `true` is returned for an alphanumeric: [a-z], [A-Z], [0-9]

See Also [“iswalnum”](#)

iswalpha

Tests for alphabetical wide characters.

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

Prototype

```
#include <wctype.h>
int iswalpha (wchar_t wc);
```

Parameters The parameter for this function is:

wc wchar_t The wide character to test

Remarks Provides the same functionality as isalpha for wide character type.

Return In the “C” locale true is returned for an alphabetic: [a-z], [A-Z]

See Also [“iswalpha”](#)

iswblank

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

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

Prototype

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

Parameters Parameters for this facility are:

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.
- See Also [“iswspace” on page 600](#)

iswcntrl

Tests for control wide characters.

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

Prototype `#include <wctype.h>
int iswcntrl (wchar_t wc);`

Parameters The parameter for this function is:

wc wchar_t The wide character to test

Remarks Provides the same functionality as iscntrl for wide character type.

Return True for the delete character (0x7F) or an ordinary control character from 0x00 to 0x1F.

See Also [“iswcntrl”](#)

iswdigit

Tests for digital wide characters.

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

Prototype `#include <wctype.h>
int iswdigit (wchar_t wc);`

Parameters The parameter for this function is:

wc wchar_t The wide character to test

Remarks Provides the same functionality as isdigit for wide character type.

Return In the “C” locale true is returned for a numeric character: [0-9].

See Also [“iswdigit”](#)

iswgraph

Tests for graphical wide characters.

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

Prototype `#include <wctype.h>
int iswgraph (wchar_t wc);`

Parameters The parameter for this function is:

wc wchar_t The wide character to test

Remarks Provides the same functionality as isgraph for wide character type.

Return In the "C" locale true is returned for a non-space printing character from the exclamation (0x21) to the tilde (0x7E).

See Also ["iswgraph"](#)

iswlower

Tests for lowercase wide characters.

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

Prototype `#include <wctype.h>
int iswlower (wchar_t wc);`

Parameters The parameter for this function is:

wc wchar_t The wide character to test

Remarks Provides the same functionality as islower for wide character type.

Return In the "C" locale true is returned for a lowercase letter: [a-z].

See Also ["iswlower"](#)

iswprint

Tests for printable wide characters.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wctype.h> int iswprint (wchar_t wc);	
Parameters	The parameter for this function is:	
	wc	wchar_t The wide character to test
Remarks	Provides the same functionality as isprint for wide character type.	
Return	In the "C" locale true is returned for a printable character from space (0x20) to tilde (0x7E).	
See Also	"iswprint"	

iswpunct

Tests for punctuation wide characters.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wctype.h> int iswpunct (wchar_t wc);	
Parameters	The parameter for this function is:	
	wc	wchar_t The wide character to test
Remarks	Provides the same functionality as ispunct for wide character type.	
Return	True for a punctuation character. A punctuation character is neither a control nor an alphanumeric character.	
See Also	"iswpunct"	

iswspace

Tests for whitespace wide characters.

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

Prototype `#include <wctype.h>
int iswspace (wchar_t wc);`

Parameters The parameter for this function is:

 wc wchar_t The wide character to test

Remarks Provides the same functionality as isspace for wide character type.

Return In the “C” locale true is returned for a space, tab, return, new line, vertical tab, or form feed.

See Also [“isspace”](#)

iswupper

Tests for uppercase wide characters.

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

Prototype `#include <wctype.h>
int iswupper (wchar_t wc);`

Parameters The parameter for this function is:

 wc wchar_t The wide character to test

Remarks Provides the same functionality as isupper for wide character type.

Return In the “C” locale true is returned for an uppercase letter: [A-Z].

See Also [“isupper”](#)

iswxdigit

Tests for a hexadecimal wide character type.

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

Prototype `#include <wctype.h>
int iswxdigit(wchar_t wc);`

Parameters	The parameter for this function is:	
	wc	wchar_t The wide character to test
Remarks	Provides the same functionality as isxdigit for wide character type.	
Return	True for a hexadecimal digit [0-9], [A-F], or [a-f].	
See Also	“isxdigit”	

towctrans

Maps a wide character type to another wide character type.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wchar.h> wint_t towctrans(wint_t c, wctrans_t value);	
Parameters	Parameters for this function are:	
	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.	
Return	Returns the remapped character.	
See Also	“wctrans” on page 603	

towlower

Converts wide characters from upper to lowercase.

Compatibility	This function may not be implemented on all platforms. Please refer to http://www.metrowerks.com/docs/MSLCompatibility for a Compatibility list.	
Prototype	#include <wctype.h> wchar_t towlower (wchar_t wc);	
Parameters	Parameters for this function are:	

	wc	wchar_t	The wide character to convert
Remarks	Provides the same functionality as tolower for wide character type.		
Return	The lowercase equivalent of a uppercase letter and returns all other characters unchanged		
See Also	"tolower" "towupper"		

towupper

Converts wide characters from lower to uppercase.

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

Prototype

```
#include <wctype.h>
wchar_t towupper (wchar_t wc);
```

Parameters The parameter for this function is:

wc wchar_t The wide character to convert

Remarks Provides the same functionality as toupper for wide character type.

Return The uppercase equivalent of a lowercase letter and returns all other characters unchanged.

See Also ["toupper"](#)
["towlower"](#)

wctrans

Constructs a property value for "toupper" and "tolower" for character remapping.

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

Prototype

```
#include <wchar.h>
wctrans_t wctrans (const char *name);
```

wctype.h*Overview of wctype.h*

Parameters	Parameters for this function are:		
	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 <code>toupper</code> or <code>tolower</code> .		
Return	A <code>wctrans_t</code> type		
See Also	“towctrans” on page 602		

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 605](#) A description of SIOUX properties.
- [“WinSIOUX for Windows” on page 606](#) 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” on page 27](#), 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.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	
------	------	----------	--------	---------	-------	--

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.

See Also [“Overview of console.h” on page 49.](#)

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-sizable, 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 607](#) basic steps to create a WinSIOUX program.
- [“Customizing WinSIOUX” on page 607](#) settings used to create the WinSIOUX console
- [“clrscr” on page 608](#) 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.

Table 45.1 The SIOUXSettings structure

This field...		Specifies...
char	initializeTB	Not applicable to WinSIOUX.
char	standalone	Not applicable to WinSIOUX.
char	setupmenus	Not applicable to WinSIOUX.
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.

Compatibility	Windows compatible, This function is not be implemented directly on Windows console.
Prototype	#include <WinSIOUX.h> void clrscr(void);
Parameters	None
Remarks	This function simply calls the function WinSIOUXclrscr, that clears the screen.

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 611
- “ [MSL IMP EXP](#) ” on page 612
- “ [MSL INTEGRAL MATH](#) ” on page 612
- “ [MSL MALLOC 0 RETURNS NON NULL](#) ” on page 612
- “ [MSL OS DIRECT MALLOC](#) ” on page 613
- “ [MSL CLASSIC MALLOC](#) ” on page 613
- “ [MSL USE NEW FILE APIS](#) ” on page 613
- “ [MSL USE OLD FILE APIS](#) ” on page 614
- “ [MSL POSIX](#) ” on page 614
- “ [SET_ERRNO](#) ” on page 614

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 Flags

Overview of the MSL Switches, Flags and Defines

_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 `UseDLLPrefix.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 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

MSL Flags

Overview of the MSL Switches, Flags and Defines

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.

--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.

MSL Flags

Overview of the MSL Switches, Flags and Defines

Index

Symbols

_ANSI_OVERLOAD_ 611
_path2fss 262
_SET_ERRNO_ 614
_tbyname 54
_beginthread 235
_beginthreadex 236
_chdrive 89
_chsizE 90
_clrscr 40
_creat 124
_CRTStartup 59
_DllTerminate 58
_dup 503
_dup2 504
_endthread 237
_endthreadex 237
_fcntl 125
_fcreator 523
_filelength 90
_fileno 316
_findclose 162
_finddata_t 161
_findfirst 162
_findnext 163
_ftype 524
_fullpath 92
_gcvt 92
_get_osfhandle 94
_getch 40
_getche 40
_getdcwd 75
_getdiskfree 76
_getdrive 93
_getdrives 76
_gotoxy 41
_HandleTable 58
_heapmin 94
_initscr 41
_inp 41
_inpD 42
_inpW 42
_IOFBF 377
_IOLBF 377
_IONBF 377
_itoa 95
_itow 95
_kbhit 43
_ltoa 96
_ltow 96
_makepath 97
_MSL_CLASSIC_MALLOC 613
_MSL_IMP_EXP 612
_MSL_IMP_EXP_C 612
_MSL_IMP_EXP_RUNTIME 612
_MSL_IMP_EXP_SIoux 612
_MSL_INTEGRAL_MATH 612
_MSL_MALLOC_0_RETURNS_NON_NULL 612
_MSL_NEEDS_EXTRAS 613
_MSL_OS_DIRECT_MALLOC 613
_MSL_POSIX 614
_MSL_USE_NEW_FILE_APIS 613
_MSL_USE_OLD_FILE_APIS 614
_open 126
_open_osfhandle 98
_outp 43
_outpd 44
_putenv 98, 424
_RunInit 59
_searchenv 99
_SetupArgs 60
_splitpath 99
_strcmpi 101
_strdate 101
_strdup 102
_strcmp 102
_stricoll 103
_strlwr 103
_strncmpi 105
_strncoll 105
_strnicmp 106
_strnicoll 107
_strnset 108
_strrev 108
_strset 109

Index

_strspnp 109
_strupr 110
_textattr 45
_textbackground 45
_textcolor 46
_ultow 112
_wcsdup 113
_wcsicmp 113
_wcsicoll 114
_wcslwr 114
_wcsncoll 115
_wcsnicmp 116
_wcsnicoll 116
_wcsnset 117
_wcsrev 118
_wcsset 118
_wcsspnp 119
_wcsupr 119
_wherex 46
_wherey 46
_wstrrev 120
_wtoi 120

Numerics

32718
Head B
heapmin 94

A

abort 401
abs 402
access 496
acos 182
acosf 183
acosh 212
acosl 183
alloca 173
alloca 35
alloca.h 35–36
AltiVec 322, 334, 355, 371
 longjmp 240
 setjmp 241
AltiVec Extensions
 fprintf 322
 printf 355
 scanf 334, 371

ANSI C 26
Argc 57
Argv 58
asctime 478
asin 184
asinf 184
asinh 213
asinl 185
assert 37
assert.h 37–38
atan 185
atan2 186
atan2f 187
atan2l 187
atanf 185
atanh 214
atanl 185
atexit 403
atof 405
atoi 406
atoi 407
atol 407
atol 407

B

bsearch 408
btowc 543

C

calloc 412
ccommand 50
ceil 187
ceilf 188
ceill 188
cerr 253
CHAR_BIT 167
CHAR_MAX 167
CHAR_MIN 167
chdir 497
chmod 270
chsize 90
cin 253
clearerr 301
clock 479
clock_t 476

close 499
closedir 79
clrscr 51, 608
Command-line Arguments 49
console.h 49–55
copysign 215
cos 188
cosf 189
cosh 189
coshf 190
coshl 190
cosl 189
cout 253
creat 124
creat 124
crtl.h 57–60
ctime 481
ctype.h 61–73
cuserid 502
Customizing SIOUX 255
Customizing WinSIOUX 607

D

Data Types
 floating point environment 131
Date and time 476
DBL_DIG 146
DBL_EPSILON 146
DBL_MANT_DIG 145
DBL_MAX 146
DBL_MAX_10_EXP 146
DBL_MAX_EXP 146
DBL_MIN 146
DBL_MIN_10_EXP 146
DBL_MIN_EXP 146
difftime 481
direct.h 75–76
dirent.h 77–79
div 414
div_t 81
div_t structure 414
dup 503
dup2 504

E

environ 58
Environment 132
EOF 300
erf 216
erfc 216
errno 83
errno.h 83–86
Error number definitions 84
exevp 506
exec 505
execle 505
execlp 505
execv 505
execve 506
exit 415
exp 190
exp2 217
expf 192
expl 192
expm1 218
extras.h 87–121

F

F_DUPFD 125
fabs 192
fabsf 193
fabsl 193
fclose 303
fcntl 125
fcntl 125
fcntl.h 123–130
fdim 219
fdopen 305
feclearexcept 133
fegetenv 141
fegetround 139
feholdexcept 141
fenv.h 131–144
FENV_ACC 133
fenv_t 131
feof 306
fraiseexcept 136
ferror 307
fesetenv 143

Index

fesetexceptflag 136
fesetround 140
fetestexcept 137
feupdateenv 143
fexcept_t 131
fflush 309
fgetc 311
fgetpos 313
fgets 315
fgetwc 543
fgetws 544
FILE 297
File Mode Macros
 Non Windows 269
 Windows 270
File Modes 269
filelength 90
fileno 91
float.h 145
Floating Point Classification Macros 179
Floating Point Math Facilities 182
Floating point mathematics 178
Floating-Point Exception Flags 132
floor 193
floorf 194
floorl 194
FLT_DIG 146
FLT_EPSILON 146
FLT_MANT_DIG 145
FLT_MAX 146
FLT_MAX_10_EXP 146
FLT_MAX_EXP 146
FLT_MIN 146
FLT_MIN_10_EXP 146
FLT_MIN_EXP 146
FLT_RADIX 145
FLT_ROUNDS 145
fmax 219
fmin 220
fmod 194
fmodf 195
fmodl 195
fopen 316
fpclassify 180, 181, 182
fprintf 319
fputc 327
fputs 328
fputwc 545
fputws 545
fread 330
free 416
freopen 332
frexp 196
frexpf 197
freexpl 197
fscanf 333
fseek 340
fsetpos 342
FSp_fopen 147
FSp_fopen.h 147–149
FSRef_fopen 148
fstat 271
ftell 343
fwide 344
fwprintf 546
fwrite 346
fwscanf 547

G

gamma 221
gcvt 92, 444
getc 347
getch 40, 52
getchar 349
getche 40
getcwd 507
getegid 509
getenv 417
geteuid 509
getgid 509
GetHandle 93
getlogin 508
getpgrp 509
getpid 509
getpid 508
getppid 509
gets 350
getuid 509
getwc 548
getwchar 548
gmtime 483

H

HUGE_VAL 212

HUGE_VAL 178

hypot 222

I

imaxabs 155

imaxdiv 155

imaxdiv_t 152

inp 41

inp_d 42

Input Control String 334

Input Conversion Specifiers 334

inp_w 42

InstallConsole 52

INT_MAX 168

INT_MIN 168

INT16_C 293

INT32_C 293

INT64_C 293

INT8_C 293

Integral limits 167

INTMAX_C 293

Intrinsic functions 28

Introduction 23–29

inttypes.h 151–160

io.h 161–163

isalnum 63

isalpha 65

isatty 510

isblank 65

iscntrl 66

isdigit 66

isfinite 181

isgraph 67

isgreater 197

isgreaterless 197

isless 198

islessequal 198

islower 67

isnan 181

iso646.h 165

isprint 68

ispunct 69

isspace 69

isunordered 199

isupper 70

iswalnum 596

iswalpha 597

iswblank 597

iswcntrl 598

iswdigit 598

iswgraph 599

iswlower 599

iswprint 599

iswpunct 600

iswspace 600

iswupper 601

iswxdigit 601

isxdigit 71

itoa 95, 444

itow 95, 444

J

jmp_buf 239

K

kbhit 43, 52

L

labs 418

LC_ALL 171

LC_COLLATE 171

LC_CTYPE 171

LC_MONETARY 171

LC_NUMERIC 171

LC_TIME 171

lconv structure 169

LDBL_DIG 146

LDBL_EPSILON 146

LDBL_MANT_DIG 145

LDBL_MAX 146

LDBL_MAX_10_EXP 146

LDBL_MAX_EXP 146

LDBL_MIN 146

LDBL_MIN_10_EXP 146

LDBL_MIN_EXP 146

ldexp 199

ldexpf 200

ldexpl 200

Index

ldiv 419
ldiv_t 81
ldiv_t structure 419
lgamma 223
limits.h 167
llabs 419
lldiv 420
lldiv_t 82
LLONG_MAX 168
LLONG_MIN 168
Locale specification 169
locale.h 169–171
localeconv 170
localtime 484
log 200
log10 202
log10f 203
log10l 203
log1p 223
log2 224
logb 225
logf 202
logl 202
LONG_MAX 168
LONG_MIN 168
longjmp 240
lseek 511
ltoa 96, 444

M

Mac OS X
 Extras Library 28
Macros
 floating point environment 131
makepath 97, 444
malloc.h 173–174
Marco Piovanelli 253
math.h 175–234
MB_LEN_MAX 168
mblen 421
mbstate_t 542
mbstowcs 422
mbtowc 423
memchr 446
memcmp 448

memcpy 449
memmove 450
memset 451
mkdir 272
mktime 484
modf 203
modff 204
modfl 204
MSL Extras Library 27
 Mac OS X 28
MSL Flags 611–615
Multithreading 31–33

N

NaN 178
nan 226
nearbyint 227
nextafter 227
NULL 285

O

offsetof 285
open 126
 open 126
opendir 77
outp 43
outpd 44
Output Control String 320
Output Conversion Specifiers 320
outpw 44

P

path2fss 262
 perror 352
POSIX
 naming conventions 27
pow 204
powf 206
powl 206
printf 353
process.h 235–237
ptrdiff_t 286
putc 361
putchar 363
putenv 98

puts 364
putwc 552
putwchar 553

Q

qsort 424
Quiet 178

R

raise 249
rand 425
RAND_MAX 426
read 512
ReadCharsFromConsole 53
readdir 78
realloc 427
remainder 229
remove 365
RemoveConsole 53
remquo 230
rename 366
rewind 368
rewinddir 78
rint 230
rinttol 231
rmdir 513
round 232
Rounding Directions 132
roundtol 232

S

scalb 233
scanf 369
Scanset 337
SCHAR_MAX 167
SCHAR_MIN 167
SEEK_CUR 340
SEEK_END 340
SEEK_SET 340
setbuf 375
setjmp 241
setjmp.h 239–241
setlocale 171
setvbuf 376
SHRT_MAX 168

SHRT_MIN 168
SIG_DFL 247
SIG_ERR 247
SIG_IGN 247
SIGABRT 246, 401
SIGBREAK 246
SIGFPE 246
SIGILL 246
SIGINT 246
signal 247
Signal handling 245
signal.h 245–250
Signaling 178
SIGSEGV 247
SIGTERM 247
sin 206
sinf 207
sinh 207
sinhf 208
sinhl 208
sinl 207
SIOUX 27, 252
SIOUX.h 251–265
SIOUXHandleOneEvent 263
SIOUXSettings structure 256, 608
size_t 286
sleep 515
snprintf 378
spawn 517
spawnl 517
spawnle 517
spawnlp 517
spawnlpe 518
spawnnv 517
spawnve 517
spawnvp 517
spawnvpe 518
splitpath 99, 444
sprintf 379
sqrt 208
sqrtf 209
sqrtl 209
srand 428
sscanf 381
Standard definitions 285

Index

Standard input/output 297
stat 273
Stat Structure
 Macintosh 268
stat.h 267–276
stdarg.h 277–281
stdbool.h 283
stddef.h 285–286
stderr 297
stdin 253, 297
stdint.h 287–293
stdio.h 295–397
stdlib.h 399–444
stdout 253, 297
strcasecmp 100
strcat 451
 strchr 452
strcmp 453
strcmpl 101
strcoll 171, 454
strcpy 456
strcspn 457
strdate 101
strupr 102, 471
Stream Orientation 300, 540
Streams 297
strerror 458
strftime 485
strcmp 102, 471
strcoll 103
string.h 445–472
strlen 459
strlwr 103, 471
strncasecmp 104
strncat 104
strncmp 461
strncpy 105
strncoll 105
strncpy 462
strnicmp 106, 471
strnicoll 107
strnset 108, 471
strpbrk 464
strrchr 465
strrev 108, 471
strset 109, 471

strspn 465
strspnp 109
strstr 467
strtoimax 156
strtok 468
strtol 431
strtold 434
strtoul 436
strtoumax 157
strupr 110, 471
strxfrm 470
swprintf 553
swscanf 554
system 439

T

tan 209
tanf 210
tanh 211
tanhf 212
tanhl 212
tanl 211
tell 110
tgmath.h 473
time 491
time.h 475–493
time_t 476
timeval structure 531
Tm Structure Members. 477
tmpfile 382
tmpnam 383
tolower 71
toupper 72
towlower 602
towupper 603
trunc 233
ttynname 518
tzname 478
tzset 492

U

UCHAR_MAX 167
UINT16_C 293
UINT32_C 293
UINT64_C 293

UINT8_C 293
 UINTMAX_C 293
 ULLONG_MAX 168
 ULONG_MAX 168
 ultoa 444
 uname 533
 ungetc 385
 Unicode 300, 541
 unistd.h 495–521
 unix.h 110–111, 523–525
 unlink 519
 USHRT_MAX 168
 Using SIOUX 251
 Using WinSIOUX 605
 utime 527
 utime.h 527–531
 utimes 530
 utsname structure 534
 utsname.h 533–534

V

va_arg 278
 va_copy 278
 va_end 279
 va_list 277
 va_start 279
 Variable arguments 277
 vec_calloc 439
 vec_free 440
 vec_malloc 441
 vec_realloc 441
 vfprintf 386
 vfscanf 389
 vfwprintf 558
 vfwscanf 555
 vprintf 391
 vsnprintf 392
 vsprintf 394
 vsprintf 394
 vsscanf 396
 vswprintf 559
 vswscanf 556
 vwprintf 560
 vwscanf 557

W

WASTE 253
 watof 561
 wchar.h 537–594
 WCHAR_MAX 542
 WCHAR_MIN 542
 wchar_t 286
 wcscat 562
 wcschr 563
 wcsncmp 563
 wcsccoll 564
 wcsncpy 565
 wcsccpnp 564
 wcsdup 113, 472, 594
 wcsftime 566
 wcsicmp 113, 472, 594
 wcsicoll 114
 wcsincmp 472, 594
 wcslen 566
 wcslwr 114, 472, 594
 wcsncat 567
 wcsncmp 568
 wcsncoll 115
 wcsncpy 568
 wcsnet 117
 wcsnicmp 116
 wcsnicoll 116
 wcsnset 472, 594
 wcspbrk 569
 wcspnp 472, 594
 wcsrchr 569
 wcsrev 118, 472, 594
 wcsset 118, 472, 594
 wcsspn 571
 wcsspnp 119
 wcsstr 572
 wcstod 572
 wcstoiimax 158
 wcstok 574
 wcstombs 442
 wcstoumax 159
 wcsupr 119, 472, 594
 wcsxfrm 582
 wctime 582
 wctob 583

Index

wctomb 443
wctrans 603
wctrans_t 596
wctype.h 595–604
wctype_t 596
WEOF 543
win_t 543
WinSIOUX 606
WinSIOUX.h 605–609
wmemchr 583
wmemcmp 584
wmemcpy 585
wmemmove 585
wmemset 586
wprintf 586
write 520
WriteCharsToConsole 54
scanf 590
wstrrev 120
wtoi 444, 472, 594