CodeWarrior®

MSL C Reference

Because of last-minute changes to CodeWarrior, some of the information in this manual may be inaccurate. Please read the Release Notes on the CodeWarrior CD for the latest up-to-date information.

Revised: 990501 rdl

How to Contact Metrowerks:

U.S.A. and international	Metrowerks Corporation P.O. Box 334 Austin, TX 78758 U.S.A.
Canada	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
Ordering	Voice: (800) 377–5416 Fax: (512) 873–4901
World Wide Web	`http://www.metrowerks.com`
Registration information	`register@metrowerks.com`
Technical support	`cw_support@metrowerks.com`
Sales, marketing, & licensing	`sales@metrowerks.com`
CompuServe	goto `Metrowerks`

Table of Contents

1 Introduction

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

CodeWarrior Year 2000 Compliance

The Products provided by Metrowerks under the License agreement process dates only to the extent that the Products use date data provided by the host or target operating system for date representations used in internal processes, such as file modifications. Any Year 2000 Compliance issues resulting from the operation of the Products are therefore necessarily subject to the Year 2000 Compliance of the relevant host or target operating system. Metrowerks directs you to the relevant statements of Microsoft Corporation, Sun Microsystems, Inc., Apple Computer, Inc., and other host or target operating systems relating to the Year 2000 Compliance of their operating systems. Except as expressly described above, the Products, in themselves, do not process date data and therefore do not implicate Year 2000 Compliance issues.

For additional information, visit: `http://www.metrowerks.com/about/y2k.html`.

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 "Overview of alloca.h" on page 25 covers the non-ANSI `alloca()` function for dynamic allocation from the stack.

The "Overview of assert.h" on page 27 covers the ANSI C exception handling macro `assert()`.

The "Overview of console.h" on page 29 covers Macintosh console routines.

The "Overview of crtl.h" on page 39,covers Win32 console routines.

The "Overview of ctype.h" on page 43 covers the ANSI character facilities.

The "Overview of div_t.h" on page 57, covers two arrays for math routines.

The "Overview of errno.h" on page 59 covers ANSI global error variables.

The "Overview of fcntl.h" on page 63 covers non-ANSI control of files.

The "Overview of float.h" on page 73 covers ANSI floating point type limits,

The "Overview of FSp_fopen.h" on page 75, contains Macintosh file opening routines.

The "Overview of io.h" on page 77, contains common Windows stream input and output routines.

The "Overview of limits.h" on page 85 covers ANSI integral type limits.

The "Overview of locale.h" on page 87 covers ANSI character sets, numeric and monetary formats.

The "Overview of malloc.h" on page 91, covers the alloca function for Windows.

The "Overview of math.h" on page 93 covers ANSI floating point math facilities.

The "Overview of path2fss.h" on page 153, covers extra Macintosh file routines.

The "Overview of Process.h" on page 155, covers Windows thread process routines.

The "Overview of setjmp.h" on page 159 covers ANSI means used for saving and restoring a processor state.

The "Overview of signal.h" on page 165 covers ANSI software interrupt specifications.

The "Overview of SIOUX and WinSIOUX" on page 179 covers Metrowerks SIOUX and WinSIOUX console emulations.

The "Overview of stat.h" on page 199 covers non-ANSI file statistics and facilities.

The "Overview of stdarg.h" on page 207 covers ANSI custom variable argument facilities.

The "Overview of stddef.h" on page 213 covers the ANSI Standard Definitions.

The "Overview of stdio.h" on page 215 covers ANSI standard input and output routines.

The "Overview of stdlib.h" on page 307 covers common ANSI library facilities.

The "Overview of string.h" on page 345 covers ANSI null terminated character array facilities.

The "Overview of time.h" on page 381 covers ANSI clock, date and time conversion and formatting facilities.

The "Overview of unistd.h" on page 401 covers many of the common non-ANSI facilities.

The "Overview of unix.h" on page 425 covers some Metrowerks non-ANS facilities.

The "Overview of utime.h" on page 433 covers non-ANSI file access time facilities.

The "Overview of utsname.h" on page 439 covers the non-ANSI equipment naming facilities.

The "Overview of wchar.h" on page 443 covers the wide character set for single and array facilities.

The "Overview of wctype.h" on page 481 covers the wide character set type comparison facilities.

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. The functions, variables and macros available in this library can be used transparently by both C and C++ programs.

- `unix.h`, `unistd.h`, `stat.h`, `fcntl.h` and `utsname.h` declare several functions common on UNIX systems that are not part of the ANSI standard.

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.

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 Windows95 and Windows NT console applications. You may want to check the headers `io.h, crtl.h` and `process.h` for specific Windows console routines.

Compatibility

Each standard function has a compatibility section that indicates the operating system(s) and/or chip(s) with which the function is compatible. A sample compatibility table appears here.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

- Compatible targets are in black text
- Incompatible targets appear in grey

- Blank cells may appear in the table in support of future targets
- ANSI represents the American National Standards Institute Programming Language C / X3.159.1989 document.
- EMB/RTOS represents Embedded Real Time Operating Systems. The currently supported systems are
  - PowerPC embedded processors using the PPC EABI (Embedded Application Binary Interface)
  - The Sony PlayStation operating system
- Mac represents the Apple Macintosh operating system on either PowerPC or 68K processors
- Palm OS represents the 3Com Palm OS operating system
- Win32 represents Windows95, Windows98 and WindowsNT operating systems on x86 processors

CodeWarrior for Palm OS does not include C libraries as binary code. The library sources and headers are provided for exposition.

If you are reading a printed version of this manual as it appears in the *Inside CodeWarrior* series, you should be aware that new targets may become available after this manual goes to print.

Information about your target may not appear in this version of the printed documentation. In that case, you should consult the electronic documentation or release notes for your product to determine whether a particular function is compatible with your target.

2

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 25 that allocates memory from the stack

alloca

Description Allocates memory quickly on the stack.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

NOTE: The function alloca() is defined in malloc.h for Win32 headers.

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.

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.

NOTE: The alloca function does not apply to all embedded/ RTOS systems Please read the release notes.

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 321

"free" on page 326

"malloc" on page 329

"realloc" on page 335)

3

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 27, that outputs a diagnostic message and stops the program if a test fails.

assert

Description Abort a program if a test is false.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 308.

Listing 3.1 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
*/
```

4

console.h

This header file contains one function, `ccommand`, which helps you port a program that relies on command-line arguments.

Overview of console.h

This header file contains one function

- "ccommand" on page 30, which helps you port a program that relies on command-line arguments.
- "clrscr" on page 33, clears the SIOUX window and flushes the buffer.
- "getch" on page 33 returns the keyboard character pressed when an ascii key is pressed
- "InstallConsole" on page 34 installs the Console package.
- "kbhit" on page 34 returns true if any keyboard key is pressed without retrieving the key
- "ReadCharsFromConsole" on page 35 reads from the Console into a buffer.
- "RemoveConsole" on page 35 removes the console package.
- "__ttyname" on page 36 Returns the name of the terminal associated with the file id. The unix.h function ttyname calls this function
- "WriteCharsToConsole" on page 37 writes a stream of output to the Console window.

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

ccommand

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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

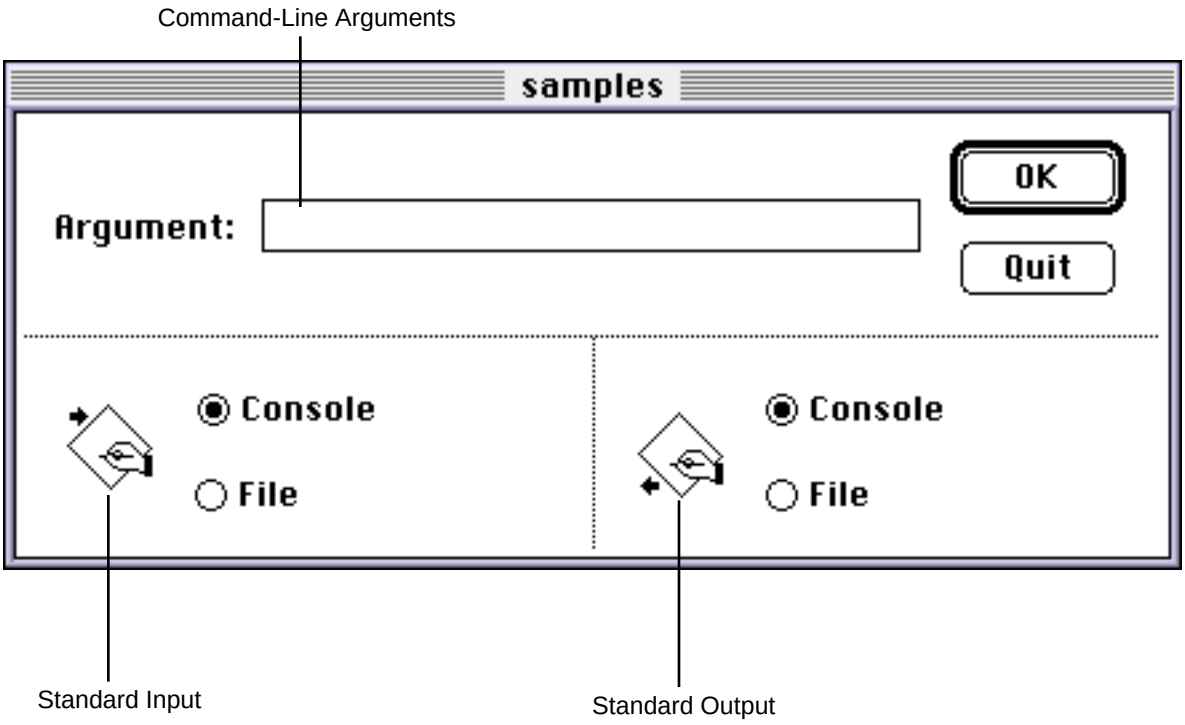
WARNING! The function ccommand() must be the first code generated in your program. It must directly follow any variable declarations in the main function.

Remarks This function displays a dialog that lets you enter arguments and redirect standard input and output, as shown in "The ccommand dialog" on page 31. Please refer to "Overview of SIOUX and WinSIOUX" on page 179, 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.

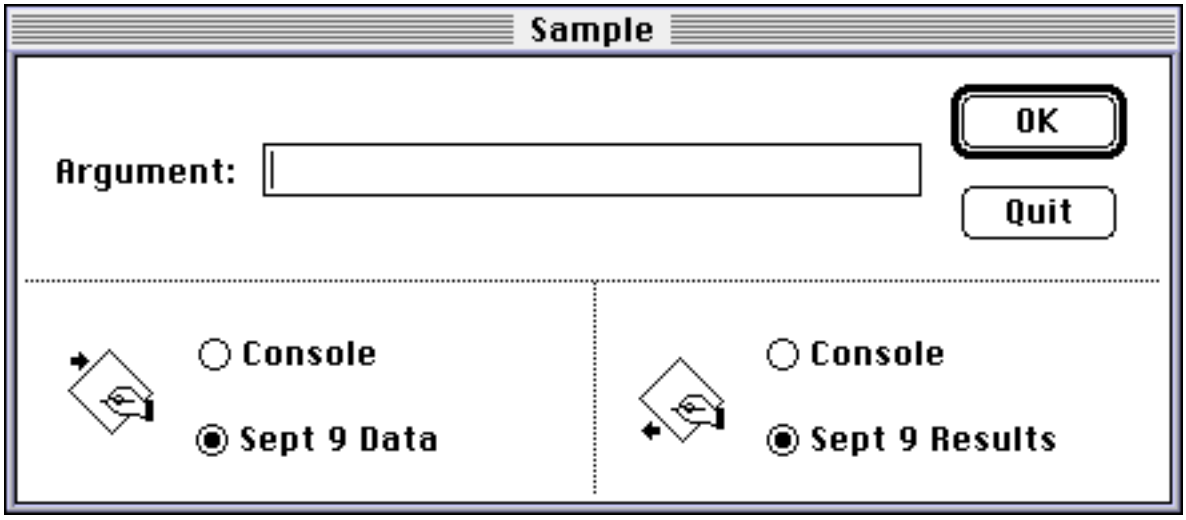
The maximum number of arguments that can be entered is determined by the value of MAX_ARGS defined in ccommand.c and is set to 25. Any arguments in excess of this number are ignored.

Figure 4.1 The ccommand dialog



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*, as shown in "Redirecting input and output to files" on page 32.

Figure 4.2 Redirecting input and output to files



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 32 has an example of command line usage

Return This function returns the number of arguments you entered.

See Also "Customizing SIOUX" on page 186

Listing 4.1 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

Description Clears the console window and flushes the buffers;

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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.

See Also "SIOUXclrscr" on page 194

getch

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 34

InstallConsole

Description Installs the Console package.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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 186
"RemoveConsole" on page 35

kbhit

Description Returns true if any keyboard key is pressed.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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 33

ReadCharsFromConsole

Description Reads from the Console into a buffer.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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 37

RemoveConsole

Description Removes the console package.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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 atexit or __atexit) 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 186

"InstallConsole" on page 34

__ttyname

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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 420

WriteCharsToConsole

Description Writes a stream of output to the Console window.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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 35

5

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 39, is the argument list count
- "Argv" on page 40, the argument list variables
- "__DllTerminate" on page 40, shows when a DLL is running terminate code.
- "environ" on page 40, is the environment pointers
- "__HandleTable" on page 41, is a structure allocated for each ed file handle
- "__CRTStartup" on page 41, initializes the C Runtime start-up routines.
- "__RunInit" on page 42, initializes the runtime, static classes and variables.
- "__SetupArgs" on page 42, sets up the command line arguments.

Argc

Description The argument count variable

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

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

Remarks Used for command line argument count.

Argv

Description The argument command variables.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

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

Remarks The command line arguments.

_DllTerminate

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

```
#include <crtl.h>
```

extern int _DllTerminate;

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

environ

Description The environment pointers

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

```
#include <crtl.h>
```

```
extern char *(*environ);
```

Remarks This is a pointer to the environment.

_HandleTable

Description FileStruct is a structure allocated for each file handle

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters None

_RunInit

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

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

Parameters None

_SetupArgs

Description The function _SetupArgs sets up the command line arguments.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

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

Parameters None

6

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 44 tests for alphabetical and numerical characters
- "isalpha" on page 46 tests for alphabetical characters
- "iscntrl" on page 47 tests for control characters
- "isdigit" on page 47 tests for digit characters
- "isgraph" on page 48 tests for graphical characters
- "islower" on page 49 tests for lower case characters
- "isprint" on page 49 tests for printable characters
- "ispunct" on page 50 tests for punctuation characters
- "isspace" on page 51 tests for white space characters
- "isupper" on page 51 test for upper case characters
- "isxdigit" on page 52 texts for hexadecimal characters
- "tolower" on page 53 changes from uppercase to lowercase

- "toupper" on page 54 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 44 for return values.

Table 6.1 Character testing functions

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

isalnum

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 45.

Return Table 6.1 describes what the character testing functions return.

See Also "tolower" on page 53

"toupper" on page 54

Listing 6.1 Character testing functions example

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

int main(void)
{
  int a = 'F', b = '6', c = '#', d = 9;

  printf("isalnum for %c: %d\n", b, isalnum(b));
  printf("isalpha for %c: %d\n", a, isalpha(a));
  printf("iscntrl for %c: %d\n", d, iscntrl(d));
  printf("isdigit for %c: %d\n", d, isdigit(d));
  printf("isgraph for %c: %d\n", d, isgraph(d));
  printf("islower for %c: %d\n", a, islower(a));
  printf("isprint for %c: %d\n", d, isprint(d));
  printf("ispunct for %c: %d\n", c, ispunct(c));
  printf("isspace for %c: %d\n", d, isspace(d));
  printf("isupper for %c: %d\n", b, isupper(b));
```

```
  printf("isxdigit for %c: %d\n", a, isxdigit(a));

  return 0;
}
```

```
Output:
isalnum for 6: 32
isalpha for F: 2
iscntrl for : 64
isdigit for : 0
isgraph for : 0
islower for F: 0
isprint for : 0
ispunct for #: 8
isspace for : 64
isupper for 6: 0
isxdigit for F: 1
```

isalpha

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 44 describes what the character testing functions return.

Listing 6.2 For example usage

```
For example usage see “Character testing functions example” on
page 45
```

iscntrl

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 44 describes what the character testing functions return.

Listing 6.3 For example usage

```
For example usage see “Character testing functions example” on
page 45
```

isdigit

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 44 describes what the character testing functions return.

Listing 6.4 For example usage

```
For example usage see "Character testing functions example" on
page 45
```

isgraph

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 44 describes what the character testing functions return.

Listing 6.5 For example usage

```
For example usage see “Character testing functions example” on
page 45
```

islower

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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`.

Return “Character testing functions” on page 44 describes what the character testing functions return.

Listing 6.6 For example usage

```
For example usage see “Character testing functions example” on
page 45
```

isprint

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 44 describes what the character testing functions return.

Listing 6.7 For example usage

```
For example usage see "Character testing functions example" on
page 45
```

ispunct

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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`.

Return "Character testing functions" on page 44 describes what the character testing functions return.

Listing 6.8 For example usage

```
For example usage see “Character testing functions example” on
page 45
```

isspace

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `c`.

Return “Character testing functions” on page 44 describes what the character testing functions return.

Listing 6.9 For example usage

```
For example usage see “Character testing functions example” on
page 45
```

isupper

Description Determine character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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`.

Return "Character testing functions" on page 44 describes what the character testing functions return.

Listing 6.10 For example usage

```
For example usage see "Character testing functions example" on
page 45
```

isxdigit

Description Determine hexidecimal type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 45

Return "Character testing functions" on page 44 describes what the character testing functions return.

Listing 6.11 For example usage

```
For example usage see “Character testing functions example” on
page 45
```

tolower

Description Character conversion macro.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this facility are:

c	int	character being evaluated

Remarks The `tolower()` macro converts an uppercase letter to its lowercase equivalent. Non-uppercase characters are returned unchanged. For example usage see “Example of tolower(), toupper() usage.” on page 53.

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

See Also “isalpha” on page 46

“toupper” on page 54.

Listing 6.12 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

Description Character conversion macro.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this facility are:

c	int	character being evaluated

Remarks The `toupper()` macro converts a lowercase letter to its uppercase equivalent and returns all other characters unchanged.

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

See Also "isalpha" on page 46
"tolower" on page 53

Listing 6.13 For example usage

```
see "Example of tolower(), toupper() usage." on page 53
```

7

div_t.h

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

Overview of div_t.h

The div_t.h header file consists of two structures.

- "div_t" on page 57, stores remainder and quotient variables
- "ldiv_t" on page 57, stores remainder and quotient variables

div_t

Description Stores the remainder and quotient from the `div` function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

See Also "div" on page 323

ldiv_t

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

See Also "ldiv" on page 328

8

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 59

errno

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

WARNING! The math library used for PowerPC 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 fully portable) provides a better error reporting mechanism. 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.

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 60,

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 8.1

Table 8.1 Error number definitions

errno value	Description
EDOM	Domain error. The arguments passed to the function are not within a legal domain. .
ERANGE	Range error. The function cannot return a value represented by its type.
ENOERR	No Error is equal to zero
EPOS	Error in stream position
ESIGPARM	Error Signal Paameter
nonzero value	Used by some standard C functions.
Win32 Only	**Description**
EPERM	Permission Error
EACCES	Permission denied
EBADF	Bad file number
EDEADLOCK	Resource deadlock will not occur
EMFILE	Too many files opened
ENOENT	No such file or directory
ENFILE	No File

ENOSPC	No space left on device
EINVAL	Invalid argument
EIO	Error on input or output
ENOMEM	Not enough memory
ENOSYS	Error no system

BeOS Only	**Description**
EOK	EOK is equal to `ENOERR`

Listing 8.1 errno example

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

int main(void)
{
  double x, y, result;

  printf("Enter two floating point values.\n");
  scanf("%lf %lf", &x, &y);
  errno = 0;// reset errno before doing operation
  result = pow(x, y);

  if (errno == EDOM)
    printf("Domain error!\n");
  else
    printf("%f to the power of %f is %f.\n",x, y, result);

  return 0;
}
```

```
/* Output:
Enter two floating point values.
1.2
3.4
```

```
1.200000 to the power of 3.400000 is 1.858730.
*/
```

9

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 63 for creating a file
- "fcntl" on page 65 for file control descriptor
- "open" on page 67 for opening a file
- "umask" on page 70 sets file permission mask

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.

creat

Description Create a new file or overwrite an existing file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <fcntl.h>
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 "fopen" on page 235
"fdopen" on page 428
"open" on page 67
"close" on page 404.

Listing 9.1 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

Description Manipulates a file descriptor.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <fcntl.h>
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.

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

See Also "fileno" on page 429
"open" on page 67
"fdopen" on page 428.

Listing 9.2 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

Description `Opens a file and returns it's id.`

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <fcntl.h>
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 `open()` opens a file for system level input and output. and is used with the UNIX style functions `read`() and `write`().

Table 9.1 Legal file opening modes with open()

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.

Mode	Description
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).
F_DUPFD	Return a duplicate file descriptor.

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

See Also "close" on page 404
"lseek" on page 416
"read" on page 417
"write" on page 423.

Listing 9.3 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\none of those country\
 gentlemen, who adorn their\nhalls with rusty lance\
 and worm-eaten targets.";

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

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

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

  /*move back to over write ... characters */
  if( lseek( fdes, -3L, SEEK_CUR ) == -1L)
  {
    printf("Seek Error");
    exit( EXIT_FAILURE );
  }

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

  /* move to beginning of file for read */
  if( lseek( fdes, 0L, SEEK_SET ) == -1L)
  {
```

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

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

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

   puts(temp);

  return 0;
}
```

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

umask

Description Sets a UNIX style file creation mask.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	**Win32**	

Prototype

```
#include <fcntl.h>
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.

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.

See Also "creat" on page 63
"open" on page 67
"mkdir" on page 202

10

float.h

The `float.h` header file macros specify the "Floating point number characteristics" on page 73 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 74

Compatibility

This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

"Floating point characteristics" on page 74 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 towards positive infinity.

Table 10.1 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 FLT_RADIX 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 FLT_RADIX can be raised to and still be expressible.
FLT_MAX_10_EXP, DBL_MAX_10_EXP, LDBL_MAX_10_EXP	The largest positive integer exponent that 10 can be raised to and still be expressible.
FLT_MIN, DBL_MIN, LDBL_MIN	The smallest positive floating point value.
FLT_MAX, DBL_MAX, LDBL_MAX	The largest floating point value.
FLT_EPSILON, DBL_EPSILON, LDBL_EPSILON	The smallest fraction expressible.

11

FSp_fopen.h

The FSp_fopen.h header defines FSp_fopen function.

Overview of FSp_fopen.h

The FSp_fopen.h header file consist of

- "FSp_fopen" on page 75 a Macintosh file opening for fopen

FSp_fopen

Description Opens a file with FSpec and return a FILE pointer.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this facility are:

spec	`ConstFSSpecPtr`	A toolbox file pointer
open_mode	char *	The open mode

Remarks The function FSp_fopen opens a file with the Macintosh Toolbox FSpec function and return a FILE pointer.

NOTE: 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.

Return The FSp_fopen facility returns a FILE pointer

See Also "fopen" on page 235

12

io.h

The header io.h defines several Windows console functions.

Overview of io.h

The alloca.h header file consists of

- "_chdir" on page 77, changes directories
- "_chdrive" on page 78, changes drives.
- "_fileno" on page 78, returns the file handle
- "_getcwd" on page 80, reads the current working directory
- "GetHandle" on page 80, gets a device handle
- "_get_osfhandle" on page 79, gets operating system file handle
- "_heapmin" on page 81, releases unused heap to the system
- "_isatty" on page 81, determines if the device is a character device
- "_makepath" on page 82, creates a path
- "_open_osfhandle" on page 82, opens a OS file handle
- "_searchenv" on page 83, searches the environment for a file

_chdir

Description This function is used to change the directory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

```
#include <io.h>
```

```
int _chdir(const char *dirname);
```

Parameters Parameters for this function are:

dirname	const char *	The new directory name

Return `False` if successful, or sets an `errno` variable and returns `-1` if unsuccessful.

See Also "_chdrive" on page 78
"_makepath" on page 82

_chdrive

Description This function is used to change drives.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

drive	int	The drive to change to

Return `Chdrive` returns `false` if successful and `true` if unsuccessful

See Also "_chdir" on page 77
"_makepath" on page 82

_fileno

Description This function retunes the file handles ID.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

```
#include <io.h>
int _fileno(FILE *stream);
```

Parameters Parameters for this function are:

stream	FILE	The stream to find the ID of

Remarks The result is unspecified if the `stream` argument does not specify and open file.

Return The file handles ID.

See Also "_get_osfhandle" on page 79

_get_osfhandle

Description The function gets the Operating Systems file handle

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

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

Parameters Parameters for this function are:

filehandle	int	the file handle

Return The operating system file handle if successful otherwise sets `errno` and returns `NULL`.

See Also "_fileno" on page 78

"_open_osfhandle" on page 82

_getcwd

Description Gets the current working directory

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <io.h>
char * _getcwd(
   char *path,
   int maxlen);
```

Parameters Parameters for this function are:

path	`char *`	A buffer to store the string
maxlen	int	Max length of buffer

Return A pointer to the path buffer, or sets an `errno` value and returns `NULL` if unsuccessful.

See Also "_searchenv" on page 83

GetHandle

Description GetHandle retrieves the current objects handle.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters None

Return The device handle.

See Also "_isatty" on page 81

_heapmin

Description This function releases the heap memory back to the system.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters None

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

_isatty

Description This function determines if the device is a character device.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
int _isatty(int fileno);
```

Parameters Parameters for this function are:

fileno	int	The devices ID

Return True if the device is a character device otherwise false.

See Also "GetHandle" on page 80

_makepath

Description Makepath is used to create a path.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

path	char *
drive	const char *
dir	const char *
fname	const char *
ext	const char *

Return None

See Also "_chdir" on page 77
"_chdrive" on page 78

_open_osfhandle

Description Open_osfhandle opens an operating system file handle.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

ofshandle	long	The Handle to open
flags	int	mode

Return Returns the handle if successful otherwise returns -1.

See Also "GetHandle" on page 80
"_get_osfhandle" on page 79

_searchenv

Description Searchenv, searches the environment for a path.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

filename	const char *	File name to search
varname	const char *	The environment variable
pathname	char *	The file name

Return None

See Also "_getcwd" on page 80

13

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 85.

Integral type limits

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

Compatibility

This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

"Integral limits" on page 85 describes the macros.

Table 13.1 Integral limits

Macro	Description
CHAR_BIT	Number of bits of smallest object that is not a bit field.
CHAR_MAX	Maximum value for an object of type `char`.
CHAR_MIN	Minimum value for an object of type `char`.
SCHAR_MAX	Maximum value for an object of type `signed char`.

SCHAR_MIN	Minimum value for an object of type `signed char`.
UCHAR_MAX	Maximum value for an object of type `unsigned char`.
SHRT_MAX	Maximum value for an object of type `short int`.
SHRT_MIN	Minimum value for an object of type `short int`.
USHRT_MAX	Maximum value for an object of type `unsigned short int`.
INT_MAX	Maximum value for an object of type `int`.
INT_MIN	Minimum value for an object of type `int`.
LONG_MAX	Maximum value for an object of type `long int`.
LONG_MIN	Minimum value for an object of type `long int`.
ULONG_MAX	Maximum value for an object of type `unsigned long int`.

14

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 87 are:

- "lconv structure and contents returned by localeconv()" on page 87
- "localeconv" on page 88 to get the locale
- "setlocale" on page 88 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 only the `"C"` locale.

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 14.1 on page 87. An `lconv` member is assigned "CHAR_MAX" on page 85 value if it is not applicable to the current locale.

Listing 14.1 lconv structure and contents returned by localeconv()

```
struct lconv {
  char *currency_symbol;
  char *int_curr_symbol;
```

```
char *mon_decimal_point;
char *mon_grouping;
char *mon_thousands_sep;
char *negative_sign;
char *positive_sign;
char frac_digits;
char int_frac_digits;
char n_cs_precedes;
char n_sep_by_space;
char n_sign_posn;
char p_cs_precedes;
char p_sep_by_space;
char p_sign_posn;
char *decimal_point;
char *grouping;
char *thousands_sep;
};
```

localeconv

Description Return the `lconv` settings for the current locale.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Description Query or set locale information for the C compiler.

Compatibility

This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters

Parameters for this facility are:

category	int	The part of the C comiler 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 357

15

malloc.h

This header defines one function, `alloca`, which lets you allocate memory quickly on from the stack.

Overview of malloc.h

The malloc.h header file consists of:

- "alloca" on page 91 that allocates memory from the stack

alloca

Description Allocates memory quickly on the stack.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

```
#include <malloc.h>
void *alloca(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 321
"free" on page 326
"malloc" on page 329
"realloc" on page 335)

16

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 98, classifies floating point numbers
- "isfinite" on page 99, tests if a value is a finite number
- "isnan" on page 99, test if a value is a computable number
- "isnormal" on page 100, tests for normal numbers
- "signbit" on page 100, tests for a negative number

Functions

- "acos" on page 101, determines the arccosine
- "asin" on page 102, determines the arcsine
- "atan" on page 103, determines the arctangent
- "atan2" on page 104, determines the arctangent of two variables
- "ceil" on page 106, determines the smallest int not less than x
- "cos" on page 108, determines the cosine
- "cosh" on page 109, determines the hyperbolic cosine
- "exp" on page 110, computes the exponential
- "fabs" on page 112 determines the absolute value
- "floor" on page 113, determines the largest integer not greater than x
- "fmod" on page 114, determines the remainder of a division

- "frexp" on page 116, extracts a value of the mantissa and exponent
- "ldexp" on page 120, computes a value from a mantissa and exponent
- "log" on page 122, determines the natural logarithm
- "log10" on page 123, determines the logarithm to base 10
- "modf" on page 124, separates integer and fractional parts
- "pow" on page 126, raises to a power
- "sin" on page 128, determines the sine
- "sinh" on page 129, determines the hyperbolic sine
- "sqrt" on page 131, determines the square root
- "tan" on page 132, determines the tangent
- "tanh" on page 133, determines the hyperbolic tangent

C9X Implementations

- "acosh" on page 135, computes the (non-negative) arc hyperbolic cosine
- "asinh" on page 136, computes the arc hyperbolic sine
- "atanh" on page 136, computes the arc hyperbolic tangent
- "copysign" on page 137, produces a value with the magnitude of x and the sign of y
- "erf" on page 137, computes the error function
- "erfc" on page 138, complementary error function
- "exp2" on page 138, computes the base-2 exponential
- "expm1" on page 139, Computes the exponential minus 1
- "fdim" on page 139, computes the positive difference of its arguments
- "fmax" on page 140, computes the maximum numeric value of its argument
- "fmin" on page 141, computes the minimum numeric value of its arguments
- "gamma" on page 141, computes the gamma function

- "hypot" on page 142, computes the square root of the sum of the squares of the arguments
- "isgreater" on page 118, compares two numbers for x greater than y
- "isgreaterless" on page 118, compares numbers for x not equal to y
- "isless" on page 119, compares two numbers for x less than y
- "islessequal" on page 119, compares two numbers for x is less than or equal to y
- "isunordered" on page 120, compares two numbers for unorder
- "lgamma" on page 142, computes the log of the absolute value
- "log1p" on page 143, computes the natural- log of x plus 1
- "log2" on page 143, computes the base-2 logarithm
- "logb" on page 144, extracts the exponent of a double value
- "nan" on page 145, Tests for NaN
- "nearbyint" on page 145, rounds off the argument to an integral value
- "nextafter" on page 146, determines the next representable value in the type of the function
- "remainder" on page 146, computes the remainder x REM y required by IEC 559
- "remquo" on page 147, computes the same remainder as the remainder function
- "rint" on page 148, rounds off the argument to an integral value
- "rinttol" on page 148, rinttol rounds its argument to the nearest long integral value
- "round" on page 149, rounds its argument to an integral value in floating-point format
- "roundtol" on page 150, roundtol rounds its argument to the nearest integral value
- "scalb" on page 150, computes x * FLT_RADIX^n

- "trunc" on page 151, 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 "strtol" on page 339 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 8.1 on page 60) 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's the signalling and quiet. The difference between a signalling NaN and quiet NaN is that both have a full exponent and both have at least one non zero significant bit, but thesignalling NaN has it's 2 most significant bits as 1 where a quiet NaN has only the second most significant bit as 1.

Quiet NaN

A quiet NaN is the result of an indeterminatecalculation such as zero divided by zero, infinity minus infinity. The IEEE floating-point standard guarantees that quiet NaN's are detectable by requiring that the invalid exception be raised whenever a 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 a `signalling NaN`. IEEE 754 requires that in such a situation the invalid exception be raised and the `signalling NaN` be coverted 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 99, 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 96

fpclassify

Description Classifies floating point numbers.

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 99

"isnan" on page 99

"isnormal" on page 100

"signbit" on page 100

"NaN Not a Number" on page 96

Listing 16.1 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

Description The facility isfinite tests if a value is a finite number.

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 98

isnan

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

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 98
"NaN Not a Number" on page 96

isnormal

Description A test of a normal number.

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
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 98

signbit

Description A test for a number that includes a signed bit

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 98

Floating Point Math Facilities

acos

Description Arccosine function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

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

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 97, for information on newer error testing procedures.

See "Example of acos(), asin(), atan(), atan2() usage." on page 105 for example usage.

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 97, for information on newer error testing procedures.

See Also "Inlined Intrinsics Option" on page 97

"cos" on page 108

"errno" on page 59

acosf

Implements the acos() function for float type values. See "acos" on page 101.

acosl

Implements the acos() function for long double type values. See "acos" on page 101.

asin

Description Arcsine function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

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 97, for information on newer error testing procedures.

See "Example of acos(), asin(), atan(), atan2() usage." on page 105 for example usage.

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 97, for information on newer error testing procedures.

See Also "Inlined Intrinsics Option" on page 97

"sin" on page 128

"errno" on page 59

asinf

Implements the asin() function for float type values. See "asin" on page 102.

asinl

Implements the asin() function for long double type values. See "asin" on page 102.

atan

Description Arctangent function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks This function computes the value of the arc tangent of the argument. See "Example of acos(), asin(), atan(), atan2() usage." on page 105 for example usage.

Return The function `atan()` returns the arc tangent of the argument `x` in the range

[-π/2, +π/2] radians.

See Also "tan" on page 132

"errno" on page 59

atanf

Implements the atan() function for float type values. See "atan" on page 103.

atanl

Implements the atan() function for long double type values. See "atan" on page 103.

atan2

Description Arctangent function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks This function computes the value of the tangent of x/y using the sines of both arguments. See "Example of acos(), asin(), atan(), atan2() usage." on page 105 for example usage.

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

[-π, ++ radians.

See Also "Inlined Intrinsics Option" on page 97

"tan" on page 132

"errno" on page 59

Listing 16.2 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 104.

atan2l

Implements the atan2() function for long double type values. See "atan2" on page 104.

ceil

Description Compute the smallest floating point number not less than *x*.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 113
"fmod" on page 114
"round" on page 149

Listing 16.3 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 106.

ceill

Implements the ceil() function for long double type values. See "ceil" on page 106.

cos

Description Compute cosine.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 128
"tan" on page 132

Listing 16.4 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 108.

cosl

Implements the cos() function for long double type values. See "cos" on page 108.

cosh

Description Compute the hyperbolic cosine.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 97

"sinh" on page 129

"tanh" on page 133

Listing 16.5 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 109.

coshl

Implements the cosh() function for long double type values. See "cosh" on page 109.

exp

Description Compute e^x.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 97
"log" on page 122
"expm1" on page 139
"exp2" on page 138
"pow" on page 126

Listing 16.6 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 110.

expl

Implements the exp() function for long double type values. See "exp" on page 110.

fabs

Description Compute the floating point absolute value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 113
"ceil" on page 106
"fmod" on page 114

Listing 16.7 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 112.

fabsl

Implements the fabs() function for long double type values. See "fabs" on page 112.

floor

Description Compute the largest floating point not greater than *x*.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

See Also "ceil" on page 106
"fmod" on page 114
"fabs" on page 112

Listing 16.8 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 113.

floorl

Implements the floor() function for long double type values. See "floor" on page 113.

fmod

Description Return the floating point remainder of x / y.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 113
"ceil" on page 106
"fmod" on page 114
"fabs" on page 112

Listing 16.9 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 114.

fmodl

Implements the fmod() function for long double type values. See "fmod" on page 114.

frexp

Description Extract the mantissa and exponent.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks 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.

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 120
"fmod" on page 114

Listing 16.10 frexp() example

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

int main(void)
{
  double m, value = 12.0;
  int e;

  m = frexp(value, &e);

  printf("%f = %f * 2 to the power of %d.\n",value, m, e);

  return 0;
}
```

```
Output:
12.000000 = 0.750000 * 2 to the power of 4.
```

frexpf

Implements the frexp() function for float type values. See "frexp" on page 116.

frexpl

Implements the frexp() function for long double type values. See "frexp" on page 116.

isgreater

Description The facility determine the greater of two doubles

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this facility are:

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

Remarks Unlike `x>y` `isgreater` does not raise an invalid exception when x and y are unordered.

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

isgreaterless

Description The facility determines if two numbers are unequal.

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks Unlike `x>y || x<y` `isgreaterless` does not raise an invalid exception when x and y are unordered.

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

isless

Description The facility determines the lesser of two numbers.

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks Unlike `x<y` `isless` does not raise an invalid exception when x and y are unordered.

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

islessequal

Description The facility test for less than or equal to comparison.

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks Unlike `x<y || x==y islessequal` does not raise an invalid exception when x and y are unordered.

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

isunordered

Description The facility compares the order of the arguments.

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Description Compute a value from a mantissa and exponent.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks The `ldexp()` function computes `x` *2^{exp}. This function can be used to construct a `double` value from the values returned by the `frexp()` function.

Return `ldexp()` returns `x` * 2`exp`.

See Also "frexp" on page 116
"modf" on page 124

Listing 16.11 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 120.

ldexpl

Implements the ldexp() function for long double type values. See "ldexp" on page 120.

log

Description Compute the natural logarithms.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 $\log_e x$. If `x` < 0 the `log()` may assign `EDOM` to `errno` See "Floating point error testing." on page 97, for information on newer error testing procedures.

See Also "Inlined Intrinsics Option" on page 97

"exp" on page 110

"errno" on page 59

Listing 16.12 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 122.

logl

Implements the log() function for long double type values. See "log" on page 122.

log10

Description Compute the base 10 logarithms.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `log10()` returns $\log_{10}x$. If `x` < 0 `log10()` may assign `EDOM` to `errno`. See "Floating point error testing." on page 97, for information on newer error testing procedures.

See Also "Inlined Intrinsics Option" on page 97

"exp" on page 110

"errno" on page 59

Listing 16.13 For example of usage see:

"log(), log10() example" on page 123

log10f

Implements the log10() function for float type values. See "log10" on page 123.

log10l

Implements the log10() function for long double type values. See "log10" on page 123.

modf

Description Separate integer and fractional parts.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks 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`.

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 116

"ldexp" on page 120

Listing 16.14 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.
```

fmod

Implements the modf() function for float type values. See "modf" on page 124.

modfl

Implements the modf() function for long double type values. See "modf" on page 124.

pow

Description Calculate x^y.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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. The `pow()` function may assign `EDOM` to `errno` if `x` is 0.0 and `y` is less than or equal to zero or if `x` is less than zero and y is not an integer. See "Floating point error testing." on page 97, for information on newer error testing procedures.

See Also "Inlined Intrinsics Option" on page 97
"sqrt" on page 131
"Example usage of error detection" on page 99,

Listing 16.15 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 126.

powl

Implements the pow() function for long double type values. See "pow" on page 126.

sin

Description Compute sine.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Remarks The argument for the sin() function should be in radians. One radian is equal to 360/2π degrees.

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

See Also "cos" on page 108

"tan" on page 132

Listing 16.16 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 128.

sinl

Implements the sin() function for long double type values. See "sin" on page 128.

sinh

Description Compute the hyperbolic sine.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 to large.

See Also "Inlined Intrinsics Option" on page 97
"cosh" on page 109
"tanh" on page 133

Listing 16.17 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 129.

sinhl

Implements the sinh() function for long double type values. See "sinh" on page 129.

sqrt

Description Calculate the square root.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 97
"pow" on page 126

Listing 16.18 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 131.

sqrtl

Implements the sqrt() function for long double type values. See "sqrt" on page 131.

tan

Description Compute tangent.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <math.h>

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

Parameters Parameters for this function are:

x	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 97

"cos" on page 108

"sin" on page 128

Listing 16.19 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 132.

tanl

Implements the tan() function for long double type values. See "tan" on page 132.

tanh

Description Compute the hyperbolic tangent.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 109
"sinh" on page 129

Listing 16.20 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 133.

tanhl

Implements the tanh() function for long double type values. See "tanh" on page 133.

HUGE_VAL

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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.

C9X Implementations

Although not formally accepted by the ANSI/ISO committee these proposed math functions are already implemented on some platforms.

acosh

Description Acosh computes the (non-negative) arc hyperbolic cosine of x in the range [0, +INF] a domain error occurs for arguments less than 1a range error occurs if x is too large.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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.

See Also "acos" on page 101

asinh

Description Asinh computes the arc hyperbolic sine

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

x	double	The value to compute

Return The hyperbolic arcsine of the argument x.

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

See Also "asin" on page 102

atanh

Description The function atanh computes the arc hyperbolic tangent.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 103

copysign

Description The function copysign produces a value with the magnitude of x and the sign of y

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

x	double	Magnitude
y	double	The sign argument

Remarks The copysign function regards the sign of zero as positive. It produces a NaN with the sign of y if x is NaN.

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

erf

Description The function erf computes the error function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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`.

erfc

Description The function computes the complementary error function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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`.

exp2

Description The function exp2 computes the base-2 exponential.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

x	double	The value to compute

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

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

See Also "pow" on page 126

expm1

Description The function expm1 computes the base-e exponential minus 1.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

fdim

Description The function fdim computes the positive difference of its arguments

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

fmax

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 141

fmin

Description The function fmin computes the minimum numeric value of its arguments.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 140

gamma

Description The function gamma computes the gamma function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 A domain error occurs if x is equal to zero or if x is a negative integer

- A range error may occur.

hypot

Description The function hypot computes the square root of the sum of the squares of the arguments.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

x	double	The first value to be squared
y	double	The second value to be squared

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

Remarks Hypot computes the square root of the sum of the squares of x and y without undue overflow or underflow.

- A range error may occur.

See Also "Inlined Intrinsics Option" on page 97

lgamma

Description The function `lgamma` computes the log of the absolute value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

x	double	The value to be computed

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

Remarks May create a range error occurs if x is too large

log1p

Description The function log1p computes the base-e logarithm.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 122

log2

Description The function log2 computes the base-2 logarithm.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 122

logb

Description The function logb extracts the exponent of a double value

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

x	double	The value being computed

Return The exponent of x as a signed integral value in the format of the x argument.

Remarks If x is subnormal it is treated as though it were normalized. A range error may occur if x is equal to zero.

See Also "Inlined Intrinsics Option" on page 97

nan

Description The function nan tests for NaN.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 96, fore more information.

See Also "isnan" on page 99

"NaN Not a Number" on page 96

nearbyint

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 rint but doesn't raise an inexact exception.

nextafter

Description 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

Compatibility This facility is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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`.

remainder

Description Remainder computes the remainder `x REM y` required by IEC 559.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

x	double	The first value
y	double	The second value

Return The remainder x REM y

See Also "remquo" on page 147

remquo

Description The function remquo computes the same remainder as the remainder function.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 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.

NOTE: 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 146

rint

Description The function rint rounds off the argument to an integral value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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.

Remarks Rounds its argument to an integral value in floating-point format using the current rounding direction.

See Also "rinttol" on page 148

rinttol

Description The function rinttol rounds its argument to the nearest long integral value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 Rintrol rounds its argument to the nearest integral value using the current rounding direction.

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

See Also "rint" on page 148

round

Description Round rounds its argument to an integral value in floating-point format.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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.

Remarks Rounding halfway cases away from zero, regardless of the current rounding direction

See Also "roundtol" on page 150

roundtol

Description The function roundtol rounds its argument to the nearest integral value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 Rounding halfway cases away from zero, regardless of the current rounding direction

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

See Also "round" on page 149

scalb

Description The function scalb computes x * FLT_RADIX^n.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 * FLT_RADIX^n

Remarks The function scalb computes x * FLT_RADIX^n efficiently, not normally by computing FLT_RADIX^n explicitly.

- A range error may occur

trunc

Description Trunc rounds its argument to an integral value in floating-point format.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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.

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

17

path2fss.h

This header path2fss.h defines one function, path2fss a function similar to PBMakeFSSpec.

Overview of path2fss.h

The path2fss.h header file consists of

- "path2fss" on page 153 a function similar to PBMakeFSSpec.

path2fss

Description his function is similar to PBMakeFSSpec.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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:

- 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"

18

Process.h

This header Process.h defines the threadex functions _beginthreadex and _endthreadex.

Overview of Process.h

The Process.h header file consists of

_beginthreadex

Description Begins a thread.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 156

_endthreadex

Description Exits the thread.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 Exit-Thread except this functions properly destroys the thread local data used by MSL.

See Also "_beginthreadex" on page 155

19

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 161 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 160 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 uninitialized `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

Description Restore the processor state saved by `setjmp()`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 setjmp()

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.

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.

WARNING! 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 161
"signal" on page 168
"abort" on page 308

Listing 19.1 For example of long jmp() usage

"setjmp() example" on page 161.

setjmp

Description Save the processor state for `longjmp()`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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()`.

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 160

"signal" on page 168

"abort" on page 308

Listing 19.2 setjmp() example

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

```
#include <stdlib.h>

// Let main() and doerr() both have
// access to global env
volatile jmp_buf env;

void doerr(void);
int main(void)
{

  int i, j, k;

  printf("Enter 3 integers that total less than 100.\n");
  printf("A zero sum will quit.\n\n");

  // If the total of entered numbers is not less than 100,
  // program execution is restarted from this point.

  if (setjmp(env) != 0)
    printf("Try again, please.\n");

  do {
    scanf("%d %d %d", &i, &j, &k);
    if ( (i + j + k) == 0)
      exit(0);// quit program
    printf("%d + %d + %d = %d\n\n", i, j, k, i+j+k);
    if ( (i + j + k) >= 100)
      doerr(); // error!
  } while (1);// loop forever

  return 0;
}

void doerr(void)// this is the error handler
{
  printf("The total is >= 100!\n");
  longjmp(env, 1);
}
Output:
```

```
Enter 3 integers that total less than 100.
A zero sum will quit.

10 20 30
10 + 20 + 30 = 60

-4 5 1000
-4 + 5 + 1000 = 1001

The total is >= 100!
Try again, please.
0 0 0
```

20

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. Additional semantics hold for the BeOS implementation see "Be Specific Signal Handling" on page 171.

- These signals, described in "signal.h Signal descriptions" on page 167, are defined as macros in the `signal.h` file.
- "signal" on page 168 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 170 calls the signal handling function
- "Signal function handling arguments" on page 167 describes the pre-defined signal handling macros that expand to functions.

BeOS has added functionality

- "sigaction" on page 174, a signal action
- "sigprocmask" on page 174, a procedure mask
- "sigpending" on page 175, a signal is pending
- "sigsuspend" on page 175, suspends a signal
- "kill" on page 176, kills a signal
- "send_signal" on page 176, sends a signal
- "struct vregs" on page 177, a structure for signal handlers

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 170, and, in the case of the `SIGABRT` signal, through the function "abort" on page 308. 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 20.1 signal.h Signal descriptions

Macro	Description
SIGABRT	Abort signal. This macro is defined as a positive integer value. This signal is called by the `abort()` function.
SIGFPE	Floating point exception signal. This macro is defined as a positive integer value.
SIGILL	Illegal instruction signal. This macro is defined as a positive integer value.
SIGINT	Interactive user interrupt signal. This macro is defined as a positive integer value.
SIGSEGV	Segment violation signal. This macro is defined as a positive integer value.
SIGTERM	Terminal signal. This macro is defined as a positive integer value. When raised this signal terminates the calling program by calling the 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 167 describes the pre-defined signal handling macros.

Table 20.2 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 `signal()` to designate that a signal be ignored.

Macro	Description
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 `SIG_IGN` and `SIG_DFL` as a function pointer. This value is returned when signal() cannot honor a request passed to it.

signal

Description Set signal handling

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `signal()` function returns a pointer to a signal handling routine that takes an `int` value argument.

The sig argument is the signal number associated with the signal handling function. The signals defined in `signal.h` are listed in "signal.h Signal descriptions" on page 167.

The func argument is the signal handling function. This function is either programmer-supplied or one of the pre-defined signal handlers described in "Signal function handling arguments" on page 167.

When it is raised, a signal handler's execution is preceded by the invocation of `signal(sig, SIG_DFL)`. This call to `signal()` effectively disables the user's handler. It can be reinstalled by placing a call within the user handler to `signal()` with the user's handler as its function argument.

Return `signal()` returns a pointer to the signal handling function set by the last call to `signal()` for signal sig. If the request cannot be honored, `signal()` returns `SIG_ERR`.

See Also "raise" on page 170
"abort" on page 308
"atexit" on page 311
"exit" on page 324

Listing 20.1 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

Description Raise a signal.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <signal.h>
int raise(int sig);
```

Parameters Parameters for this facility are:

sig	int	A signal handling function

Remarks The `raise()` function calls the signal handling function associated with signal sig.

Return `raise()` returns a zero if the signal is successful; it returns a non-zero value if it is unsuccessful.

See Also "longjmp" on page 160
"signal" on page 168
"abort" on page 308
"atexit" on page 311
"exit" on page 324

Listing 20.2 For example of raise() usage

```
Refer to the example for "Example of signal() usage" on page 169
```

Be Specific Signal Handling

The Posix interface for signal handling functions isn't as useful as it could be. The standard indicates that only a single argument (the signal number) is passed to the signal handler. It is useful to have more information and the BeOS provides two extra arguments.

Table 20.3 BeOS specific signal defines

Macro	Description
SIGHUP	hang-up -- tty is gone!
SIGQUIT	quit' special character typed in tty
SIGCHLD	child process exited
SIGPIPE	write to a pipe w/no readers

Macro	Description
SIGKILL	kill a team (not catchable)
SIGSTOP	suspend a thread (not catchable)
SIGCONT	continue execution if suspended
SIGTSTP	stop' special character typed in tty
SIGALRM	an alarm has gone off (see alarm())
SIGTTIN	read of tty from bg process
SIGTTOU	write to tty from bg process
SIGUSR1	app defined signal 1
SIGUSR2	app defined signal 2
SIGKILLTHR	be specific: kill just the thread, not team

"BeOS signal function handling arguments" on page 172, describes the pre-defined BeOS signal handling macros.

Table 20.4 BeOS signal function handling arguments

Macro	Description
sigemptyset	Empty set
sigfillset	Fill set
sigaddset	Add set
sigdelset	Delete set
sigismember	Is a member

For BeOS we declare the sa_handler field of the sigaction struct as type __signal_func_ptr. That means you'll need to cast any function you assign to the sa_handler field.

NOTE: C++ member functions can not be signal handlers (because they expect a “this” pointer as the first argument).

The 3 arguments that the BeOS provides to signal handlers are as follows:

- The first argument is the signal number (as an integer).
- The next argument is whatever value is put in the sa_userdata field of the sigaction struct.
- The last argument is a pointer to a vregs struct.

The vregs struct contains the contents of the volatile registers at the time the signal was delivered to your thread. You can change the fields of the structure. After your signal handler completes, the OS uses this struct to reload the registers for your thread (privileged registers are not loaded of course). The vregs struct is of course terribly machine dependent and is guaranteed to change, potentially even between different models of the PowerPC family. If you use it, you should expect to have to re-work your code when new processors come out. Nonetheless the ability to change the registers does open some interesting programming possibilities.

```
struct sigaction {
  __signal_func_ptr sa_handler;
  sigset_t          sa_mask;
  int               sa_flags;
  void             *sa_userdata;
                /*passed to the signal handler*/
};
```

Table 20.5 BeOS signal flags

Macro	Description
SIG_NOCLDSTOP	for sa_flags
SIG_BLOCK	defines for the how arg of sigprocmask()
SIG_UNBLOCK	Unblock
SIG_SETMASK	Set mask

sigaction

Description The function `sigaction` is the signal action.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <signal.h>
int sigaction(
   int sig,
   const struct sigaction *act,
   struct sigaction *oact);
```

Parameters Parameters for this function are:

sig	int	A signal
act	const struct sigaction *	A signal action
oact	struct sigaction *	A signal action

Return An Integral value.

sigprocmask

Description The function `sigprocmask` is the signal procedure mask.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <signal.h>
int sigprocmask(
   int how,
   const sigset_t *set,
   sigset_t *oset);
```

Parameters None

Parameters for this function are:

how	int	How
set	const sigset_t *	Set
oset	sigset_t *	Oset

Return An Integral value.

sigpending

Description The function `sigpending` is for a signal pending.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <signal.h>
int sigpending(sigset_t *set);
```

Parameters Parameters for this function are:

set	sigset_t *	The set

Return An Integral value.

sigsuspend

Description The function `sigsuspend` denotes a suspended signal.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <signal.h>
int sigsuspend(const sigset_t *mask);
```

Parameters Parameters for this function are:

mask	const sigset_t *	A signal set mask

Return An Integral value.

kill

Description The function `kill`, ends a signal.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

Parameters Parameters for this function are:

pid	pid_t
sig	int

Return An Integral value.

send_signal

Description The function `send_signal` sends a signal.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <signal.h>
int send_signal(pid_t tid, uint sig);
```

Parameters Parameters for this function are:

tid	pid_t
sig	uint

Return An Integral value.

struct vregs

Description Signal handlers get this as the last argument.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <signal.h>
typedef struct vregs
  {
    ulong pc,            /* program counter */
      r0,                       /* scratch */
      1,                      /* stack ptr */
      r2,                           /* TOC */
                          /* volatile regs */
      r3, r4,r5,r6,r7,r8,r9,r10,
                           /* scratch regs */
      r11, r12;

    double f0,                 /* fp scratch */

                         /* fp volatile regs */
      f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13;

    ulong filler1,             /* place holder */
    fpscr,              /* fp condition codes */
      ctr, xer, cr, msr, lr;  /* misc. status */
  }vregs;
```

21

SIOUX & WinSIOUX

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 SIOUX and WinSIOUX

In the following section the Macintosh hosted interface is known as SIOUX and the Windows hosted as WinSIOUX. The facilities and structure members for the Standard Input Output User eXchange console interface are:

"Using SIOUX and WinSIOUX" on page 179 A description of SIOUX properties.

- "WinSIOUX for Windows" on page 180 the (Simple Input and Output User eXchange) library for Windows 95 and Windows NT
- "SIOUX for Macintosh" on page 183 the (Simple Input and Output User eXchange) library for the Macintosh Operating Systems.

Using SIOUX and WinSIOUX

Sometimes you need to port a program that was originally written 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 SIOUX and 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. 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 29.

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 in the File menu, you can print or save the contents of the SIOUX window.

- "Creating a Project with WinSIOUX" on page 181 basic steps to create a WinSIOUX program.
- "Customizing WinSIOUX" on page 181 settings used to create the WinSIOUX console
- "WinSIOUXclrscr" on page 182, is used to clear the WinSIOUX console screen and buffer
- "clrscr" on page 183 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:

- ANSICx86.LIB
- ANSICx86sd.LIB

NOTE: WinSIOUX is incomplete in the current release

Customizing WinSIOUX

This following sections describe how you can customize the WinSIOUX environment by modifying the structure `tSIOUXBuffer`. WinSIOUX examines the data fields of `tSIOUXBuffer` to determine how to create the WinSIOUX window and environment.

NOTE: To customize WinSIOUX, you must modify `tSIOUXBuffer` before you call any function that uses standard input or output. If you modify `tSIOUXBuffer` afterwards, WinSIOUX does not change its window.

Table 21.1 The tSIOUXBuffer Structure

Type	Element	Purpose
char *	startpos	The pointer to a block of memory that will serve as the buffer
char *	curtop	The pointer to start of line at top of screen
char *	endpos	the pointer to end of text in the buffer
char *	inputstart	The pointer to a block of memory as keyboard input buffer

Type	Element	Purpose
char *	inputcur	The pointer to next available character of input
char *	inputlast	The pointer to character after last input character
char *	SelBasePtr	The pointer to where selection began, may be start or end
char *	SelStartPtr	The pointer to start of selected text
char *	SelEndPtr	The pointer to end of selected text
int	row	The row index of current insert point
int	maxrow	The maximum number of rows
int	col	The column index of current insert point
int	maxcol	The maximum number of columns
int	installed	Is true if console has been installed
int	inputavail	Is true if input is available for program
int	dirtybit	Is true if the buffer changed since last saving
int	NeedInput	Is true if characters are to be stored at the caret position
int	CmdShow	
int	numlines	The number of text lines in buffer
long	bufsize	The current buffer size

WinSIOUXclrscr

Description Clears the WinSIOUX window and flushes the buffers.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

```
#include <WinSIOUX.h>
void WinSIOUXclrscr(void);
```

Parameters None

Remarks This function is used to clear the console and WinSIOUX buffer.

See Also "SIOUXclrscr" on page 194

clrscr

Description Clears the WinSIOUX window and flushes the buffers.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

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

Parameters None

Remarks This function simply call WinSIOUXclrscr.

See Also "WinSIOUXclrscr" on page 182

SIOUX for Macintosh

SIOUX for Macintosh contains the following segments.

- "Creating a Project with SIOUX" on page 185 shows a running SIOUX program.
- "Customizing SIOUX" on page 186 shows how to customize your SIOUX window.

  - "The SIOUXSettings structure" on page 187 list structure members that may be set for altering SIOUX's appearance
- "Using SIOUX windows in your own application" on page 193 contains information for using Mac OS facilities with in your SIOUX project.
  - "SIOUXHandleOneEvent" on page 194 allows you to use an even in SIOUX
  - "SIOUXSetTitle" on page 195 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 191

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 192.

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 186.

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 193.

If you want to add SIOUX to a project you already created, the project must contain certain libraries.

A 68K project must contain at least these libraries:

- MSL SIOUX.68K.Lib
- MacOS.Lib

- MSL Runtime68k.lib
- MathLib suitable for your 68k project version
- MSL C.Lib suitable for your 68k project version

A PPC project must contain at least these libraries:

- MSL SIOUX.PPC.Lib
- InterfaceLib
- MSL RuntimePPC.lib
- MathLib
- MSL C.PPC.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 189, "Changing the size and location" on page 190, and "Showing the status line" on page 192, describe how to customize the SIOUX window. The next section, "Changing what happens on quit" on page 191, describe how to modify how SIOUX acts when you quit it. The last section, "Using SIOUX windows in your own application" on page 193, describes how you can use a SIOUX window in your own Macintosh program.

"The SIOUXSettings structure" on page 187 summarizes what's in the `SIOUXSettings` structure.

Table 21.2 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.
short	fontid	The font in the SIOUX window.

This field…		Specifies…
short	fontsize	The size of the font in the SIOUX window.
short	fontface	The style of the font in the SIOUX window.

"Example of customizing a SIOUX Window" on page 188 contains a small program that customizes a SIOUX window.

Listing 21.1 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 `tabspaces` and how to change the font with the fields `fontid`, `fontsize`, and `fontface`.

NOTE: The status line in the SIOUX window writes its messages with the font specified in the fields `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 192.

To change the font in the SIOUX window, set `fontid` to one of these values:

To change the font in the SIOUX window, set `fontid` to one of these values:

- `courier where the ID is kFontIDCourier`
- `geneva where the ID is kFontIDGeneva`
- `helvetica where the ID is kFontIDHelvetica`
- `monaco where the ID is kFontIDMonaco`
- `newYork where the ID is kFontIDNewYork`
- `symbol where the ID is kFontIDSymbol`
- `times where the ID is kFontIDTimes`

By default, fontid is monaco.

To change the character style for the font, set fontface to one of these values:

- `normal`
- `bold`
- `italic`
- `underline`
- `outline`
- `shadow`
- `condense`
- `extend`

To combine styles, add them together. For example, to write text that's bold and italic, set `fontface` to `bold + italic`. By default, `fontface` is `normal`.

To change the size of the font, set `fontsize` to the size. By default, `fontsize` is `9`.

The field `tabspaces` controls how SIOUX handles tabs. If `tabspaces` is any number greater than `0`, SIOUX prints that number of spaces instead of a tab. If `tabspaces` is `0`, it prints a tab. In the SIOUX window, a tab looks like a single space, so if you are printing a table, you should set `tabspaces` to an appropriate number, such as `4` or `8`. By default, `tabspaces` is `4`.

The sample below sets the font to 12-point, bold, italic New York and substitutes 4 spaces for every tab:

```
SIOUXSettings.fontsize = 12;
SIOUXSettings.fontface = bold + italic;
SIOUXSettings.fontid = kFontIDNewYork;
SIOUXSettings.tabspaces = 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 193.

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;
```

SIOUXclrscr

Description Clears the SIOUX window and flushes the buffers;

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

Prototype

```
#include <SIOUX.h>
void SIOUXclrscr(void);
```

Parameters None

Remarks This function is used to clear the console and SIOUX buffer.

See Also "WinSIOUXclrscr" on page 182

SIOUXHandleOneEvent

Description Handles an event for a SIOUX window.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

Prototype

```
#include <SIOUX.h>
Boolean SIOUXHandleOneEvent(EventRecord *event);
```

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 21.2 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

Description To set the title of the `SIOUX` output window.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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.

WARNING! The argument for `SIOUXSetTitle()` is a pascal string, not a C style string.

Return

There is no return value from `SIOUXSetTitle()`

Listing 21.3 Example of `SIOUXSetTitle()` usage.

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

int main(void)
{
  printf("Hello World\n");
  SIOUXSetTitle("\pMy Title");

  return 0;
}
```

22

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
- "fstat" on page 201 to get information on an open file
- "mkdir" on page 202 to make a directory for folder
- "stat" on page 204 to get statistics of a file

stat.h and UNIX Compatibility

The header file `unix.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 Macintosh Toolbox.

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

Stat Structure and Definitions

The header stat.h includes the `stat` structure, several type definitions and file mode definitions. Among the necessary types are

Table 22.1 Defined types

Type	Used to Store
nlink_t	The number of links
uid_t	The user's ID
gid_t	The file size
off_t	The file size in bytes

Table 22.2 Stat Structure

Type	Variable	Purpose
mode_t	st_mode	File mode, see "File Modes,"
ino_t	st_ino	File serial number
dev_t	st_dev	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
time_t	st_atime	Time of last access
time_t	st_mtime	Time of last data modification
time_t	st_ctime	Time of last file status change

Table 22.3 File Modes

File Mode	Purpose
S_IFMT	File type mask
S_IFDIR	Directory
S_IFCHR	Character special
S_IFIFO	Pipe

File Mode	Purpose
S_IFREG	Regular
S_IREAD	Read permission, owner
S_IWRITE	Write permission, owner
S_IEXEC	Execute/search permission, owner

fstat

Purpose Gets information about an open file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stat.h>
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" on page 200.

Return If it is successful, `fstat()` returns zero. If it encounters an error, `fstat()` returns `-1` and sets `errno`.

See Also "stat" on page 204

Listing 22.1 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

Purpose Makes a folder.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stat.h>
int mkdir(const char *path, int mode);
int mkdir(const char *path);
```

Parameters Parameters for this facility are:

path	const char *	The path name
mode	int	The open mode (Not applicable for Windows)

Remarks This function creates the new folder specified in `path`. It ignores the argument `mode`.

Return If it is successful, `mkdir()` returns zero. If it encounters an error, `mkdir()` returns `-1` and sets `errno`.

See Also "unlink" on page 421

"rmdir" on page 418

Listing 22.2 Example for mkdir()

```
Macintosh
```

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

int main(void)
{
  if( mkdir(":Akbar", 0) == 0)
      printf("Folder Akbar is created");

  return 0;
}
```

Windows

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

int main(void)
{
  if( mkdir(".\Akbar") == 0)
      printf("Folder Akbar is created");

  return 0;
}
```

Creates a folder named Akbar as a sub-folder of the current folder

stat

Purpose Gets information about a file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stat.h>
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" on page 200.

Return If it is successful, `stat()` returns zero.

See Also "fstat" on page 201
"uname" on page 439

Listing 22.3 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
```

23

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 208 a variable argument list
- "va_end" on page 208 a variable argument end
- "va_start" on page 209 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 209 initializes the `va_list` variable and is called before gaining access to the arguments. The macro "va_arg" on page 208 returns each of the arguments in `va_list`. When all the arguments have been processed through `va_arg()`, the macro "va_end" on page 208 is called to allow a normal return from the function.

va_arg

Description Macro to return an argument value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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	A type set by the macro

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 208

"va_start" on page 209

Listing 23.1 For example of va() usage

```
Refer to the example "Example of va_start() usage." on page 210.
```

va_end

Description Prepare a normal function return.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 208

"va_start" on page 209

Listing 23.2 For example of va_end usage

```
Refer to the example "Example of va_start() usage." on page 210.
```

va_start

Description Initialize the variable-length argument list.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 208

"va_end" on page 208

Listing 23.3 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
```

24

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 213, defines NULL
- "offsetof" on page 214, is the offset of a structure's member
- "ptrdiff_t" on page 214, used for pointer differences
- "size_t" on page 214, is the return from a sizeof operation
- "wchar_t" on page 214, 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 is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

NULL

The `NULL` macro is the null pointer constant used in the Standard Library.

offsetof

The o`ffsetof(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 p`trdiff_t` type is the signed integral type used for 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 w`char_t` type is an integral type capable of holding all character representations of the ASCII character set. .

25

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 219 clears an error from a stream
- "fclose" on page 221 closes a file
- "fdopen" on page 223, converts a file descriptor to a stream
- "ferror" on page 226 checks a file error status
- "fflush" on page 228 flushes a stream
- "fgetc" on page 229 gets a character from a file
- "fgetpos" on page 231 gets a file position from large files
- "fgets" on page 233 gets a string from a file
- "fopen" on page 235 opens a file
- "fprintf" on page 238 prints formatted output to a file
- "fputc" on page 245 writes a character to a file
- "fputs" on page 246 writes a string to a file
- "fread" on page 248 reads a file
- "freopen" on page 250 reopens a file
- "fscanf" on page 252 scans a file
- "fseek" on page 256 moves to a file position

- “fsetpos” on page 259 sets a file position for large files
- “ftell” on page 260 tells a file offset
- “fwrite” on page 261 writes to a file
- “getc” on page 262 gets a character from a stream
- “getchar” on page 264 gets a character from stdin
- “gets” on page 266 gets a string from stdin
- “perror” on page 267 writes an error to stderr
- “printf” on page 269 writes a formatted output to stdout
- “putc” on page 275 writes a character to a stream
- “putchar” on page 277 writes a character to stdout
- “puts” on page 278 writes a string to stdout
- “remove” on page 279 removes a file
- “rename” on page 281 renames a file
- “rewind” on page 282 resets the file indicator to the beginning
- “scanf” on page 284 scans stdin for input
- “setbuf” on page 288 sets the buffer size for a stream
- “setvbuf” on page 290 sets the buffer size and buffering scheme for a stream
- “sprintf” on page 292 to write to a character buffer
- “sscanf” on page 293 to scan a string
- “tmpfile” on page 295 to create a temporary file
- “ungetc” on page 298 to place a character back in a stream
- “vfprintf” on page 300 write variable arguments to file
- “vprintf” on page 302 write variable arguments to stdout
- “vsprintf” on page 304 write variable arguments to a char array buffer

Standard input/output

Streams

A stream 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`, `stdout`, `stderr` streams are line buffered.

For more information on routing `stdin`, `stdout`, and `stderr` to a Macintosh console window, see the chapter on SIOUX.h

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 un-oriented after that stream is 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.

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.

clearerr

Description Clear a stream's end-of-file and error status.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 224
"ferror" on page 226
"fopen" on page 235

"fseek" on page 256
"rewind" on page 282

Listing 25.1 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

Description Close an open file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 a -1 if it fails to close a file.

See Also
"fopen" on page 235
"freopen" on page 250
"tmpfile" on page 295
"abort" on page 308
"exit" on page 324

Listing 25.2 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;
}
```

fdopen

Description Converts a file descriptor to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
FILE *fdopen(int fildes, char *mode);
```

NOTE: fdopen() for the Macintosh is defined in the unix.h header.

Parameters Parameters for this facility are:

fildes	int	A file descriptor
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 429

"open" on page 67

Listing 25.3 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

Description Check the end-of-file status of a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 return zero if the stream is not at the end-of-file.

See Also "clearerr" on page 219
"ferror" on page 226

Listing 25.4 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

Description Check the error status of a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 219
"feof" on page 224

Listing 25.5 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

Description Empty a stream's buffer to its host environment.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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.

The fflush() function should not be used after an input operation.

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return `fflush()` returns a nonzero value if it is unsuccessful and returns zero if it is successful.

See Also "setvbuf" on page 290

Listing 25.6 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\n", count);
    if (count % 10)
      fflush(f);// flush buffer every 10 numbers
  }
  fclose(f);

  return 0;
}
```

fgetc

Description Read the next character from a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `int`. If the end-of-file has been reached, `fgetc()` returns `EOF`.

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 218

"getc" on page 262

"getchar" on page 264

Listing 25.7 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;
}
```

fgetpos

Description Get a stream's current file position indicator value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 postition 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 256
"fsetpos" on page 259
"ftell" on page 260

Listing 25.8 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

Description Read a character array from a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 char 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` 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 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 218

"gets" on page 266

Listing 25.9 For example of fgets() usage

```
Refer to "Example of feof() usage." on page 225 for feof().
```

fopen

Description Open a file as a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 The `fopen()` function opens a file specified by `filename`, and associates a stream with it. The `fopen()` function returns a pointer to a `FILE`. This pointer is used to refer to the file when performing I/O operations.

The mode argument specifies how the file is to be used. "Open modes for fopen()," describes the values for mode.

UPDATE MODE A file opened with an update mode ("+") is buffered. The file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the fflush() function or one of the file positioning operations (fseek(), fsetpos(), or rewind()). Similarly, a file cannot be read from and then written to without repositioning the file using one of the file positioning functions unless the last read or write reached the end-of-file.

All file modes, except the append modes (“a”, “a+”, “ab”, “ab+”) set the file position indicator to the beginning of the file. The append modes set the file position indicator to the end-of-file.

NOTE: Write modes, even if in Write and Read (w+, wb+) delete any current data in a file when the file is opened.

Table 25.1 Open modes for fopen()

Mode	Description
“r”	Open an existing text file for reading only.
“w”	Create a new text file for writing, or open and truncate an existing file
“a”	Open an existing text file, or create a new one if it does not exist, for appending. Writing occurs at the end-of-file position.
“r+”	Update mode. Open an existing text file for reading and writing (See Remarks)
“w+”	Update mode. Create a new text file for writing, or open and truncate an existing file, for writing and reading (See Remarks)
“a+”	Update mode. Open an existing text file or create a new one for reading and writing. Writing occurs at the end-of-file position (See Remarks)
“rb”	Open an existing binary file for reading only.
“wb”	Create a new binary file or open and truncate an existing file, for writing
“ab”	Open an existing binary file, or create a new one if it does not exist, and append. Writing occurs are the end-of-file.
“r+b” or “rb+”	Update mode. Open an existing binary file for reading and writing (See Remarks)

Mode	Description
"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 221

Listing 25.10 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;
}
```

fprintf

Description Send formatted text to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
int fprintf(FILE *stream,
       const char *format, ...);
```

Parameters Parameters for this facility are:

stream	FILE *	A pointer to a FILE stream
format	const char *	The format string

Remarks The `fprintf()` function writes formatted text to `stream` and advances the file position indicator. Its operation is the same as `printf()` with the addition of the `stream` argument. Refer to the description of `printf()`.

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the fflush() function or one of the file positioning operations (fseek(), fsetpos(), or rewind()).

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Output Control String and Conversion Specifiers

The `format` character array contains normal text and conversion specifications. Conversion specifications must have matching arguments in the same order in which they occur in `format`.

The various elements of the format string is specified in the ANSI standards to be in this order from left to right.

- A percent sign
- Optional flags -,+,0,# or space
- Optional minimum field width specification
- Optional precision specification
- Optional size specification
- Conversion operator c,d,e,E,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. "Format modifier types for formatted output functions" on page 240 describes the flag characters.

The optional minimum width is a decimal digit string. If the converted value has more characters that 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. "Format modifier types for formatted output functions" on page 240 describes the conversion type characters.

Table 25.2 Format modifier types 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.`
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.`

	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.

	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	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.

e, E	The floating point argument (`float` or `double`) is output in scientific notation: [-]*b*.*aaa*e±E*ee*. There is one digit (*b*) before the decimal point. Unless indicated by an optional precision width, the default is 6 digits after the decimal point (*aaa*). If the precision width is 0, no decimal point is output. The exponent (*ee*) is at least 2 digits long. The `e` conversion type uses lowercase `e` as the exponent prefix. The `E` conversion type uses uppercase `E` as the exponent prefix.
g, G	The `g` conversion type uses the `f` or `e` conversion types and the `G` conversion type uses the `f` or `E` conversion types. Conversion type `e` (or `E`) is used only if the converted exponent is less than -4 or greater than the precision width. The precision width indicates the number of significant digits. No decimal point is output if there are no digits following it.
c	The corresponding argument is output as a character.
s	The corresponding argument, a pointer to a character array, is output as a character string. Character string output is completed when a null character is reached. The null character is not output.
p	The corresponding argument is taken to be a pointer. The argument is output using the `X` conversion type format.
	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 218
"printf" on page 269
"sprintf" on page 292
"vfprintf" on page 300
"vprintf" on page 302
"vsprintf" on page 304

Listing 25.11 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;
}
```

fputc

Description Write a character to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 character `c` to `stream` and advances `stream`'s file position indicator. Although the `c` argument is an `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 218

"putc" on page 275

"putchar" on page 277

Listing 25.12 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;
}
```

fputs

Description Write a character array to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 218

"puts" on page 278

Listing 25.13 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;
}
```

fread

Description Read binary data from a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 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` argu-

ments 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 218

"fgets" on page 233

"fwrite" on page 261

Listing 25.14 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

Description Re-direct a stream to another file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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
moce	const char *	The file opening mode
stream	FILE *	A pointer to a FILE stream

Remarks The `freopen()` function changes the file `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 235

Listing 25.15 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

Description 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 <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 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. "Format modifier types for formatted input functions," describes the conversion type characters.

Table 25.3 Format modifier types for formatted input functions

Modifier	Description
	Size Modifiers
h	The `h` flag indicates that the corresponding conversion modifier is a `short int` or `unsigned short int` type.
l	When used with integer conversion modifiers, the `l` flag indicates `long int` or an `unsigned long int` type. When used with floating point conversion modifier, the `l` flag indicates a `double`.

ll	When used with integer conversion modifiers, the `ll` flag indicates `long long` or an `unsigned long long` type.
L	The `L` flag indicates that the corresponding float conversion modifier is a `long double` type.

	Conversion Modifiers
d	A decimal integer is read.
i	A decimal, octal, or hexadecimal integer is read. The integer can be prefixed with a plus or minus sign (+, -), 0 for octal numbers, 0x or 0X for hexadecimal numbers.
o	An octal integer is read.
u	An unsigned decimal integer is read.
x, X	A hexadecimal integer is read.
e, E, f, g, G	A floating point number is read. The number can be in plain decimal format (e.g. `3456.483`) or in scientific notation ([-]*b*`.`*aaa*e±*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 type.
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.
[scanset]	A character array is read. The *scanset* is a sequence of characters. Input stream characters are read until a character is found that is not in *scanset*. If the first character of *scanset* is a circumflex (^) then input stream characters are read until a character from *scanset* is read. A null character is appended to the end of the character array.

Return `fscanf()` returns the number of items read. 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 218

"errno" on page 59

"scanf" on page 284

Listing 25.16 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

Description Move the file position indicator.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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. Also, `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 231
"fsetpos" on page 259
"ftell" on page 260

Listing 25.17 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

Description Set the file position indicator.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 postioning type

Remarks The `fsetpos()` function sets the file position indicator for `stream` using the value pointed to by `pos`. The function is used in conjunction with `fgetpos()` when dealing with files having sizes greater than what can be represented by the `long int` argument used by `fseek()`.

`fsetpos()` undoes the previous call to `ungetc()` and clears the end-of-file status.

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return `fsetpos()` returns zero if it is successful and returns a nonzero value if it fails.

See Also "fgetpos" on page 231
"fseek" on page 256
"ftell" on page 260

Listing 25.18 For example of fsetpos() usage

```
Refer to "Example of fgetpos() usage." on page 232 for fgetpos().
```

ftell

Description Return the current file position indicator value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `ftell()` function returns the current value of stream's file position indicator. It is used in conjunction with `fseek()` to provide random access to a file.

The function will not work correctly when it is given a stream associated to a console file, such as `stdin`, `stdout`, or `stderr`, where a file indicator position is not applicable. Also, `ftell()` cannot handle files with sizes larger than what can be represented with a `long int`. In such a case, use the `fgetpos()` and `fsetpos()` functions.

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return `ftell()`, when successful, returns the current file position indicator value. If it fails, `ftell()` returns `-1L` and sets the global variable `errno` to a nonzero value.

See Also "errno" on page 59, "fgetpos" on page 231

Listing 25.19 For example of ftell() usage

```
Refer to "Example of fseek() usage." on page 258 for fseek().
```

fwrite

Description Write binary data to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size,
       size_t nmemb, FILE *stream);
```

Parameters Parameters for this facility are:

ptr	void *	A pointer to the item being written
size	size_t	The size of the item being written
nmemb	size_t	The number of elements being written
stream	FILE *	A pointer to a FILE stream

Remarks The `fwrite()` function writes `nmemb` items of `size` bytes each to `stream`. The items are contained in the array pointed to by `ptr`. After writing the array to `stream`, `fwrite()` advances the file position indicator accordingly.

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the fflush() function or one of the file positioning operations (fseek(), fsetpos(), or rewind()).

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return `fwrite()` returns the number of elements successfully written to `stream`.

See Also "Wide Character and Byte Character Stream Orientation" on page 218

"fread" on page 248

Listing 25.20 For example of fwrite() sage

```
Refer to "Example of fread() usage." on page 249 for fread().
```

getc

Description Read the next character from a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 (+) 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.

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 218

"fgetc" on page 229

"fputc" on page 245

"getchar" on page 264

"putchar" on page 277

Listing 25.21 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;
}
```

getchar

Description Get the next character from `stdin`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
int getchar(void);
```

Parameters None

Remarks The `getchar()` function reads a character from the `stdin` stream.

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 218

"fgetc" on page 229

"getc" on page 262

"putchar" on page 277

Listing 25.22 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

Description Read a character array from `stdin`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `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 218

"fgets" on page 233

Listing 25.23 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

Description Output an error message to `stderr`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
void perror(const char *s);
```

Parameters

Parameters for this facility are:

s	const char *	Prints an errno and message

Remarks

The `perror()` function outputs the character array pointed to by `s` and the value of the global variable `errno` to `stderr`.

See Also

"abort" on page 308

"errno" on page 59

Listing 25.24 Example of perror() usage.

```
#include <stdio.h>

#define MAXLIST 10

int main(void)
{
  int i[MAXLIST], count;

  printf("Enter %d numbers.\n", MAXLIST);
  printf("Numbers less than 0 will generate an error.\n");

  // read MAXLIST integer values from the console
  for (count = 0; count < MAXLIST; count++) {
    scanf("%d", &i[count]);

    // if the value is <= 0 output an error message
    // to stderr using perror()
    if (i[count] < 0)
      perror("Invalid entry!\n");
  }
  printf("Done!\n");

  return 0;
}
```

printf

Description Output formatted text.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 operator c,d,e,E,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. "Format modifier types for formatted output functions" on page 240 describes the flag characters.

The optional minimum width is a decimal digit string. If the converted value has more characters that 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.

"Format modifier types for formatted output functions" on page 240 describes the conversion type characters.

Table 25.4 Format modifier types 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.`
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.`
	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.

	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	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.
e, E	The floating point argument (`float` or `double`) is output in scientific notation: [-]*b*.*aaa*e±E*ee*. There is one digit (*b*) before the decimal point. Unless indicated by an optional precision width, the default is 6 digits after the decimal point (*aaa*). If the precision width is 0, no decimal point is output. The exponent (*ee*) is at least 2 digits long. The `e` conversion type uses lowercase `e` as the exponent prefix. The `E` conversion type uses uppercase `E` as the exponent prefix.
g, G	The `g` conversion type uses the `f` or `e` conversion types and the `G` conversion type uses the `f` or `E` conversion types. Conversion type `e` (or `E`) is used only if the converted exponent is less than -4 or greater than the precision width. The precision width indicates the number of significant digits. No decimal point is output if there are no digits following it.
c	The corresponding argument is output as a character.
s	The corresponding argument, a pointer to a character array, is output as a character string. Character string output is completed when a null character is reached. The null character is not output.
p	The corresponding argument is taken to be a pointer. The argument is output using the `X` conversion type format.

	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 218

"fprintf" on page 238

"sprintf" on page 292

"vfprintf" on page 300

"vprintf" on page 302

"vsprintf" on page 304

Listing 25.25 Example of printf() usage.

```
#include <stdio.h>

int main(void)
{
  int i = 25;
  char c = 'M';
  short int d = 'm';
  static char s[] = "Metrowerks!";
  static char pas[] = "\pMetrowerks again!";
  float f = 49.95;
  double x = 1038.11005;
  int count;
  printf("%s printf() demonstration:\n%n", s, &count);
  printf("The last line contained %d characters\n",count);
```

```
  printf("Pascal string output: %#20s\n", pas);
  printf("%-4d %x %06x %-5o\n", i, i, i, i);
  printf("%*d\n", 5, i);
  printf("%4c %4u %4.10d\n", c, c, c);
  printf("%4c %4hu %3.10hd\n", d, d, d);
  printf("$%5.2f\n", f);
  printf("%5.2f\n%6.3f\n%7.4f\n", x, x, x);
  printf("%*.*f\n", 8, 5, x);

  return 0;
}
```

```
Output:
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
```

putc

Description Write a character to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `putc()` function outputs `c` to `stream` and advances `stream`'s file position indicator.

The `putc()` works identically to the `fputc()` function, except that it is written as a macro.

If the file is opened in update mode (+) the file cannot be written to and then read from unless the write operation and read operation are separated by an operation that flushes the stream's buffer. This can be done with the fflush() function or one of the file positioning operations (fseek(), fsetpos(), or rewind()).

Return `putc()` returns the character written when successful and return `EOF` when it fails.

See Also "Wide Character and Byte Character Stream Orientation" on page 218

"fputc" on page 245

"putchar" on page 277

Listing 25.26 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;
}
```

putchar

Description Write a character to `stdout`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
int putchar(int c);
```

Parameters Parameters for this facility are:

c	int	The character to write to a 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 218

"fputc" on page 245

"putc" on page 275

Listing 25.27 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

Description Write a character string to `stdout`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 218

"fputs" on page 246

Listing 25.28 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
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
car bus metro werks
```

remove

Description Delete a file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 235

"rename" on page 281

Listing 25.29 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

Description Change the name of a file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `rename()` function changes the name of a file, specified by `old` to the name specified by `new`.

Return `rename()` returns a nonzero if it fails and returns zero if successful

See Also "freopen" on page 250
"remove" on page 279

Listing 25.30 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

Description Reset the file position indicator to the beginning of the file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 256

"fsetpos" on page 259

Listing 25.31 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

Description Read formatted text.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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. "Format modifier types for formatted input functions," describes the conversion type characters.

Table 25.5 Format modifier types for formatted input functions

Modifier	Description
	Size Modifiers

h	The `h` flag indicates that the corresponding conversion modifier is a `short int` or `unsigned short int` type.
l	When used with integer conversion modifiers, the `l` flag indicates `long int` or an `unsigned long int` type. When used with floating point conversion modifier, the `l` flag indicates a `double.`
ll	When used with integer conversion modifiers, the `ll` flag indicates `long long` or an `unsigned long long` type.
L	The `L` flag indicates that the corresponding float conversion modifier is a `long double` type.

	Conversion Modifiers
d	A decimal integer is read.
i	A decimal, octal, or hexadecimal integer is read. The integer can be prefixed with a plus or minus sign (+, -), 0 for octal numbers, 0x or 0X for hexadecimal numbers.
o	An octal integer is read.
u	An unsigned decimal integer is read.
x, X	A hexadecimal integer is read.
e, E, f, g, G	A floating point number is read. The number can be in plain decimal format (e.g. `3456.483`) or in scientific notation ([-]*b*`.`*aaa*e±*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 type.

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.
[scanset]	A character array is read. The *scanset* is a sequence of characters. Input stream characters are read until a character is found that is not in *scanset*. If the first character of *scanset* is a circumflex (^) then input stream characters are read until a character from *scanset* is read. A null character is appended to the end of the character array.

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 218

"fscanf" on page 252

"sscanf" on page 293

Listing 25.32 Example of scanf() usage.

```
#include <stdio.h>

int main(void)
{
  int i;
  unsigned int j;
  char c;
  char s[40];
  double x;

  printf("Enter an integer surrounded by ! marks\n");
  scanf("!%d!", &i);
```

```
  printf("Enter three integers\n");
  printf("in hexadecimal, octal, or decimal.\n");
  // note that 3 integers are read, but only the last two
  // are assigned to i and j
  scanf("%*i %i %ui", &i, &j);

  printf("Enter a character and a character string.\n");
  scanf("%c %10s", &c, s);

  printf("Enter a floating point value.\n");
  scanf("%lf", &x);

  return 0;
}
```

```
Output:
Enter an integer surrounded by ! marks
!94!
Enter three integers
in hexadecimal, octal, or decimal.
1A 6 24
Enter a character and a character string.
Enter a floating point value.
A
Sounds like 'works'!
3.4
```

setbuf

Description Change the buffer size of a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 290

"malloc" on page 329

Listing 25.33 Example of setbuf() usage.

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

int main(void)
{
  FILE *f;
  char name[80];

  // get a filename from the user
  printf("Enter the name of the file to write to.\n");
  gets(name);

  // create a new file for output
  if ( (f = fopen(name, "w")) == NULL) {
    printf("Can't open file %s.\n", name);
    exit(1);
  }

  setbuf(f, NULL);// turn off buffering

  // this text is sent directly to the file without
```

```
  // buffering
  fprintf(f, "Buffering is now off\n");
  fprintf(f, "for this file.\n");

  // close the file
  fclose(f);

  return 0;
}
```

```
Output:
Enter the name of the file to write to.
bufftest
```

setvbuf

Description Change the buffering scheme for a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
int setvbuf(FILE *stream, char *buf, int mode,
       size_t size);
```

Parameters Parameters for this facility are:

stream	FILE *	A pointer to a FILE stream
buf	char *	A buffer for input and output
mode	int	A buffering mode
size	size_t	The size of the buffer

Remarks The `setvbuf()` allows the manipulation of the buffering scheme as well as the size of the buffer used by stream. The function should be called after the stream is opened but before it is written to or read from.

The `buf` argument is a pointer to a character array. The `size` argument indicates the size of the character array pointed to by `buf`. The most efficient buffer size is a multiple of `BUFSIZ`, defined in `stdio.h`.

If `buf` is a null pointer, then the operating system creates its own buffer of `size` bytes.

The `mode` argument specifies the buffering scheme to be used with `stream`. `mode` can have one of three values defined in `stdio.h`: `_IOFBF`, `_IOLBF`, and `_IONBF`.

- `_IOFBF` specifies that `stream` be buffered.
- `_IOLBF` specifies that `stream` be line buffered.
- `_IONBF` specifies that `stream` be unbuffered

Return `setvbuf()` returns zero if it is successful and returns a nonzero value if it fails.

See Also "setbuf" on page 288

"malloc" on page 329

Listing 25.34 Example of setvbuf() usage.

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

int main(void)
{
  FILE *f;
  char name[80];

  // get a filename from the user
  printf("Enter the name of the file to write to.\n");
  gets(name);

  // create a new file for output
  if ( (f = fopen(name, "w")) == NULL) {
    printf("Can't open file %s.\n", name);
    exit(1);
```

```
  }

  setvbuf(f, NULL, _IOLBF, 0);// line buffering
  fprintf(f, "This file is now\n");
  fprintf(f, "line buffered.\n");

  // close the file
  fclose(f);

  return 0;
}
```

```
Output:
Enter the name of the file to write to.
buffy
```

sprintf

Description Format a character string array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `sprintf()` function works identically to `printf()` with the addition of the `s` parameter. Output is stored in the character array pointed to by `s` instead of being sent to `stdout`. The function terminates the output character string with a null character.

For specifications concerning the output control string and conversion specifiers please see: "Output Control String and Conversion Specifiers" on page 239.

Return `sprintf()` returns the number of characters assigned to `s`, not including the null character.

See Also "Wide Character and Byte Character Stream Orientation" on page 218

"fprintf" on page 238

"printf" on page 269

Listing 25.35 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

Description Read formatted text into a character string.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
int sscanf(char *s, const char *format, ...);
```

Parameters

Parameters for this facility are:

s	char *	The string to scan in to
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 sepcifications see: "Input Control String and Conversion Specifiers" on page 252.

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 218

"fscanf" on page 252

"scanf" on page 284

Listing 25.36 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

Description Open a temporary file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 235

"tmpnam" on page 296

Listing 25.37 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

Description Create a unique temporary filename.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 "fopen" on page 235

"tmpfile" on page 295

Listing 25.38 Example of tmpnam() usage.

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

int main(void)
{
  FILE *f;
  char *tempname;
  int c;

  // get a unique filename
  tempname = tmpnam("tempwerks");

  // create a new file for output
  if ( (f = fopen(tempname, "w")) == NULL) {
    printf("Can't open temporary file %s.\n", tempname);
    exit(1);
  }

  // output text to the file
```

```
  fprintf(f, "shoe shirt tie trousers\n");
  fprintf(f, "province\n");

  // close the file
  fclose(f);

  // delete the file
  remove(tempname);

  return 0;
}
```

ungetc

Description Place a character back into a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 218

"fseek" on page 256

"fsetpos" on page 259

"rewind" on page 282

Listing 25.39 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;
}
```

vfprintf

Description Write formatted output to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Listing 0.1 **Prototype**

```
#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

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 of type `va_list` processed by the `va_start()` macro from the `stdarg.h` header file.

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 239.

Return `vfprintf()` returns the number of characters written or `EOF` if it failed.

See Also "Wide Character and Byte Character Stream Orientation" on page 218

"fprintf" on page 238

"printf" on page 269

"Overview of stdarg.h" on page 207

Listing 25.40 Example of vfprintf() usage.

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

int fpr(FILE *, char *, ...);

int main(void)
{
  FILE *f;
  static char name[] = "foo";
  int a = 56, result;
  double x = 483.582;

  // create a new file for output
  if (( f = fopen(name, "w")) == NULL) {
    printf("Can't open %s.\n", name);
    exit(1);
  }

  // format and output a variable number of arguments
  // to the file
  result = fpr(f, "%10s %4.4f %-10d\n", name, x, a);

  // close the file
  fclose(f);

  return 0;
}
```

```
// fpr() formats and outputs a variable
// number of arguments to a stream using
// the vfprintf() function
int fpr(FILE *stream, char *format, ...)
{
  va_list args;
  int retval;

  va_start(args, format);// prepare the arguments
  retval = vfprintf(stream, format, args);
  // output them
  va_end(args);// clean the stack
  return retval;
}
```

vprintf

Description Write formatted output to `stdout`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdio.h>
int vprintf(const char *format, va_list arg);
```

Parameters Parameters for this facility are:

format	const char *	Teh format string
arg	va_list	A variable argument list

Remarks The `vprintf()` function works identically to the `printf()` function. Instead of the variable list of arguments that can be passed to `printf()`, `vprintf()` accepts its arguments in the array of type `va_list` processed by the `va_start()` macro from the `stdarg.h` header file.

For specifications concerning the output control string and conversion specifiers please see: "Output Control String and Conversion Specifiers" on page 239.

Return `vprintf()` returns the number of characters written or a negative value if it failed.

See Also "Wide Character and Byte Character Stream Orientation" on page 218

"fprintf" on page 238

"printf" on page 269

"Overview of stdarg.h" on page 207

Listing 25.41 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        *
```

vsprintf

Description Write formatted output to a string.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 wrte 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 239.

Return `vsprintf()` returns the number of characters written to `s` or `EOF` if it failed.

See Also "Wide Character and Byte Character Stream Orientation" on page 218

"printf" on page 269

"sprintf" on page 292

"Overview of stdarg.h" on page 207

Listing 25.42 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
```

26

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 313
- “strtod” on page 337

The pseudo-random number generation functions are

- “rand” on page 334
- “srand” on page 336

The memory management functions are

- “calloc” on page 321
- “free” on page 326
- “malloc” on page 329
- “realloc” on page 335

The environment communication functions are

- “abort” on page 308
- “atexit” on page 311
- “exit” on page 324

- "getenv" on page 326
- "system" on page 342

The searching and sorting functions are

- "bsearch" on page 316
- "qsort" on page 333

The multibyte conversion functions convert locale-specific multibyte characters to `wchar_t` type characters (defined in `stddef.h`). The functions are

- "mblen" on page 330
- "mbstowcs" on page 331
- "mbtowc" on page 332
- "wcstombs" on page 343
- "wctomb" on page 344

The integer arithmetic functions are

- "abs" on page 310
- "div" on page 323
- "labs" on page 328
- "ldiv" on page 328

Many of the `stdlib.h` functions use the `size_t` type and the `NULL` macro, which are defined in `stdlib.h`.

abort

Description Abnormal program termination.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 27, "longjmp" on page 160, "raise" on page 170, "signal" on page 168, "atexit" on page 311, "exit" on page 324

Listing 26.1 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

Description Compute the absolute value of an integer.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 112

"labs" on page 328

Listing 26.2 Example of abs() usage.

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

int main(void)
{
  int i = -20;
  long int j = -48323;

  printf("Absolute value of %d is %d.\n", i, abs(i));
  printf("Absolute value of %ld is %ld.\n", j, labs(j));

  return 0;
}
```

```
Output:
Absolute value of -20 is 20.
Absolute value of -48323 is 48323.
```

atexit

Description Install a function to be executed at a program's exit.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 324

Listing 26.3 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

Description Convert a character string to a numeric value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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`.

This function skips leading white space characters.

This function sets the global variable `errno` to `ERANGE` if the converted value cannot be expressed in their respective type.

Return `atof()` returns a floating point value of type `double`.

See Also "atoi" on page 314
"atol" on page 315
"errno" on page 59
"scanf" on page 284

Listing 26.4 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

Description Convert a character string to a numeric value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this facility are:

nptr	const char *	The character being converted

Remarks The `atoi()` 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 in their respective type.

Return `atoi()` returns an integer value of type `int`.

See Also

"atof" on page 313
"atol" on page 315
"errno" on page 59
"scanf" on page 284

atol

Description Convert a character string to a numeric value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdlib.h>

double atof(const char *nptr);
int atoi(const char *nptr);
long int atol(const char *nptr);
```

Parameters Parameters for this facility are:

nptr	const char *	The character being converted

Remarks The `atol()` function converts the character array pointed to by `nptr` to an integer of type `long int`.

This function skips leading white space characters.

This function sets the global variable `errno` to `ERANGE` if the converted value cannot be expressed in their respective type.

Return `atol()` returns an integer value of type `long int`.

See Also "atof" on page 313
"atoi" on page 314
"errno" on page 59
"scanf" on page 284

bsearch

Description Efficient sorted array searching.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 *	What you are searching for
base	const void *	The array to be searched
compare	const void *	A pointer to a function used for comparison

Remarks The `bsearch()` function efficiently searches a sorted array for an item using the binary search algorithm.

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. This function is used to compare the key with each individual element of the array. That compare function takes two 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 one element of the array passed as the second argument to `bsearch()`.

For explanation we will call the arguments search_key and array_element. This 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 333

Listing 26.5 Example of bsearch usage.

```
// A simple telephone directory manager
// This program accepts a list of names and
// telephone numbers, sorts the list, then
```

```
// searches for specified names.

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// Maximum number of records in the directory.
#define MAXDIR 40

typedef struct
{
  char lname[15];// keyfield--see comp() function
  char fname[15];
  char phone[15];
} DIRENTRY; // telephone directory record

int comp(const DIRENTRY *, const DIRENTRY *);
DIRENTRY *look(char *);
DIRENTRY directory[MAXDIR];// the directory itself
int reccount; // the number of records entered

int main(void)
{
  DIRENTRY *ptr;
  int lastlen;
  char lookstr[15];

  printf("Telephone directory program.\n");
  printf("Enter blank last name when done.\n");

  reccount = 0;
  ptr = directory;
  do {
    printf("\nLast name: ");
    gets(ptr->lname);
    printf("First name: ");
    gets(ptr->fname);
    printf("Phone number: ");
    gets(ptr->phone);
    if ( (lastlen = strlen(ptr->lname)) > 0) {
```

```
      reccount++;
      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);
    elseprintf("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

Description Allocate space for a group of objects.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdlib.h>
void *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 `calloc()` function allocates contiguous space for `nmemb` elements of size. The space is initialized with zeroes.

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

"free" on page 326
"malloc" on page 329
"realloc" on page 335

Listing 26.6 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

Description Compute the integer quotient and remainder.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 114
"ldiv" on page 328
"div_t" on page 57

Listing 26.7 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

Description Terminate a program normally.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 308

"atexit" on page 311

Listing 26.8 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

Description Release previously allocated memory to heap.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdlib.h>
void free(void *ptr);
```

Parameters Parameters for this facility are:

ptr	void *	A pointer to the allocated memory

Remarks The `free()` function releases a previously allocated memory block, pointed to by `ptr`, to the heap. The `ptr` argument should hold an address returned by the memory allocation functions `calloc()`, `malloc()`, or `realloc()`. Once the memory block `ptr` points to has been released, it is no longer valid. The `ptr` variable should not be used to reference memory again until it is assigned a value from the memory allocation functions.

See Also "calloc" on page 321
"malloc" on page 329
"realloc" on page 335

Listing 26.9 For example of free() usage

```
Refer to"Example of calloc() usage." on page 321 .
```

getenv

Description Environment list access.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdlib.h>
char *getenv(const char *name);
```

Parameters Parameters for this facility are:

name	const char *	A buffer for the environment list

Remarks For Macintosh systems the `getenv()` is an empty function that always returns a null pointer (`NULL`). It is included in the Metrowerks `stdlib.h` header file to conform to the ANSI C Standard Library specification.

Return `getenv()` returns `NULL` for the Mac. For Windows `getenv()` returns zero on failure or the environmental variable.

See Also "system" on page 342

Listing 26.10 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

Description Compute long integer absolute value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `long int` type.

See Also "fabs" on page 112
"abs" on page 310

Listing 26.11 For example of labs() usage

```
Refer to "Example of abs() usage." on page 310.
```

ldiv

Description Compute the long integer quotient and remainder.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 114
"div" on page 323
"ldiv_t" on page 57

Listing 26.12 For example of ldiv() usage

```
Refer to "Example of div() usage." on page 323 .
```

malloc

Description Allocate a block of heap memory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 "calloc" on page 321
"free" on page 326
"realloc" on page 335

Listing 26.13 For example of malloc() usage

```
Refer to "Example of calloc() usage." on page 321.
```

mblen

Description Compute the length of a multibyte character.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `mblen()` function returns the length of the multibyte character pointed to by `s`. It examines a maximum of `n` characters.

The Metrowerks C implementation supports the "C" locale only and returns the value of `mbtowc(NULL, s, n)`.

Return `mblen()` returns the value of `mbtowc(NULL, s, n)`.

See Also "Locale specification" on page 87
"mbtowc" on page 332

mbstowcs

Description Convert a multibyte character array to a `wchar_t` array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 string to convert
n	size_t	The maximum wide characters to convert

Remarks The `mbstowcs()` function converts a character array containing multibyte characters to a character array containing `wchar_t` type characters. The `wchar_t` type is defined in `stddef.h`.

The Metrowerks C implementation of `mbstowcs()` performs no translation; it copies a maximum of `n` bytes from the array pointed to by `s` to the array pointed to by `pwcs`. The function terminates prematurely if a null character is reached.

Return `mbstowcs()` returns the number of bytes copied from `s` to `pwcs`.

See Also "Locale specification" on page 87
"wcstombs" on page 343

mbtowc

Description Translate a multibyte character to a `wchar_t` type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdlib.h>
int mbtowc(wchar_t *pwc,
       const char *s, size_t n);
```

Parameters Parameters for this facility are:

pwc	wchar_t *	The wide character destination
s	const char *s	The string to convert
n	size_t	The maximum wide characters to convert

Remarks The `mbtowc()` function converts a multibyte character, pointed to by `s`, to a character of type `wchar_t`, pointed to by `pwc`. The function converts a maximum of `n` bytes.

The Metrowerks C implementation performs no translation; it copies the first character at `s` to the first character at `pwc`.

Return `mbtowc()` returns -1 if `n` is zero and `s` is not a null pointer.

`mbtowc()` returns 0 if `s` is a null pointer or s points to a null character (`'\0'`).

`mbtowc()` returns 1 if `s` is not a null pointer and it does not point to a null character (`'\0'`).

See Also "Locale specification" on page 87

"mblen" on page 330

"wctomb" on page 344

qsort

Description Sort an array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 *	A pointer to the array to be sorted
nmemb	size_t	The number of elements
size	size_t	The size of the elements
compare	void *	A pointer to a comparison function

Remarks The `qsort()` function sorts an array using the quicksort algorithm. It sorts the array without displacing it; the array occupies the same memory it had before the call to `qsort()`.

The `base` argument is a pointer to the base of the array to be sorted.

The `nmemb` argument specifies the number of array elements to sort.

The `size` argument specifies the size of an array element.

The `compare` argument is a pointer to a programmer-supplied compare function. The function takes two pointers to different array elements and compares them based on the key. If the two elements are equal, `compare` must return a zero. The `compare` function must return a negative number if the first element is less than the second. Likewise, the function must return a positive number if the first argument is greater than the second.

See Also "bsearch" on page 316

Listing 26.14 For example of qsort() usage

```
Refer to “Example of bsearch usage.” on page 317 .
```

rand

Description Generate a pseudo-random integer value.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 336

Listing 26.15 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 number 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 number for seed 1:
     16838      5758     10113     17515     31051

First five random number for seed 2:
       908     22817     10239     12914     25837

First five random number for seed 3:
     17747      7107     10365      8312     20622

First five random number for seed 4:
      1817     24166     10491      3711     15407

First five random number for seed 5:
     18655      8457     10616     31877     10193
```

realloc

Description Change the size of an allocated block of heap memory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `realloc()` function changes the size of the memory block pointed to by `ptr` to `size` bytes. The `size` argument can have a value smaller or larger than the current size of the block `ptr` points to. The `ptr` argument should be a value assigned by the memory allocation functions `calloc()` and `malloc()`.

If `size` is 0, the memory block pointed to by `ptr` is released. If `ptr` is a null pointer, `realloc()` allocates `size` bytes.

The old contents of the memory block are preserved in the new block if the new block is larger than the old. If the new block is smaller, the extra bytes are cut from the end of the old block.

Return `realloc()` returns a pointer to the new block if it is successful and `size` is greater than 0. `realloc()` returns a null pointer if it fails or `size` is 0.

See Also "calloc" on page 321
"free" on page 326
"malloc" on page 329

Listing 26.16 For example of realloc() usage

```
Refer to "Example of calloc() usage." on page 321.
```

srand

Description Set the pseudo-random number generator seed.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdlib.h>
void srand(unsigned int seed);
```

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 `seed`. Each seed value produces the same sequence of random numbers when it is used.

See Also "rand" on page 334

Listing 26.17 For example of labs() usage

```
Refer to "Example of rand() usage." on page 334.
```

strtod

Description Character array to numeric conversions.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 nptry that is not convertible.

Remarks The `strtod()` converts a character array, pointed to by `nptr`, to a floating point value of type `double`. The character array can be in

either decimal notation (e.g. `103.578`) or scientific notation (`[-]`*b*`.`*aaa*`e±E`*ee*).

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.

This function sets the global variable `errno` to `ERANGE` if there is a conversion error.

Return `strtod()` returns a floating point value of type `double`. If `nptr` cannot be converted to an expressible double value, `strtod()` returns `HUGE_VAL`, defined in `math.h`, and sets `errno` to `ERANGE`.

See Also
"strtol" on page 339
"strtoul" on page 341
"errno" on page 59
"Integral type limits" on page 85
"Overview of math.h" on page 93
"scanf" on page 284

Listing 26.18 Example of strtod(), strtol(), strtoul() usage.

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

int main(void)
{
  double f;
  long int i;
  unsigned long int j;
  static char si[] = "4733!", sf[] = "103.749?";
  static char sb[] = "0x10*";
  char *endptr;
```

```
  f = strtod(sf, &endptr);
  printf("%f %c\n", f, *endptr);

  i = strtol(si, &endptr, 10);
  printf("%ld %c\n", i, *endptr);

  i = strtol(si, &endptr, 8);
  printf("%ld %c\n", i, *endptr);

  j = strtoul(sb, &endptr, 0);
  printf("%ld %c\n", j, *endptr);


  j = strtoul(sb, &endptr, 10);
  printf("%ld %c\n", j, *endptr);

  return 0;
}
```

```
Output:
103.749000 ?
4733 !
2523 !
16 *
0 x
```

strtol

Description Character array to numeric conversions.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
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 `strtol()` function converts a character array, pointed to by `nptr`, to an integer value of type `long int`, in base. A plus or minus sign (+ or -) prefixing the number string is optional.

The `base` argument in `strtol()` and `strtoul()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. If `base` is 0, then `strtol()` and `strtoul()` convert the character array based on its format. Character arrays beginning with `'0'` are assumed to be octal, number strings beginning with `'0x'` or `'0X'` are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to the functions' respective types.

This function skips leading white space.

This function sets the global variable `errno` to `ERANGE` if there is a conversion error.

Return `strtol()` returns an integer value of type `long int`. If the converted value is less than `LONG_MIN`, `strtol()` returns `LONG_MIN` and sets `errno` to `ERANGE`. If the converted value is greater than `LONG_MAX`, `strtol()` returns `LONG_MAX` and sets `errno` to `ERANGE`. The `LONG_MIN` and `LONG_MAX` macros are defined in `limits.h`.

See Also "strtod" on page 337
"strtoul" on page 341
"errno" on page 59

"Integral type limits" on page 85
"Overview of math.h" on page 93
"scanf" on page 284

strtoul

Description Character array to numeric conversions.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
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 `strtoul()` 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 `strtol()` and `strtoul()` specifies the base used for conversion. It must have a value between 2 and 36, or 0. If `base` is 0, then `strtol()` and `strtoul()` convert the character array based on its format. Character arrays beginning with `'0'` are assumed to be octal, number strings beginning with `'0x'` or `'0X'` are assumed to be hexadecimal. All other number strings are assumed to be decimal.

If the `endptr` argument is not a null pointer, it is assigned a pointer to a position within the character array pointed to by `nptr`. This position marks the first character that is not convertible to the functions' respective types.

This function skips leading white space.

This function sets the global variable `errno` to `ERANGE` if there is a conversion error.

Return `strtoul()` returns an unsigned integer value of type `unsigned long int`. If the converted value is greater than `ULONG_MAX`, `strtoul()` returns `ULONG_MAX` and sets `errno` to `ERANGE`. The `ULONG_MAX` macro is defined in `limits.h`

See Also
"strtod" on page 337
"strtol" on page 339
"errno" on page 59
"Integral type limits" on page 85
"Overview of math.h" on page 93
"scanf" on page 284

system

Description Environment list assignment.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdlib.h>
int system(const char *string);
```

WARNING! The `system()` function is an empty function that is included in the Metrowerks `stdlib.h` to conform to the ANSI C Standard Library specification.

Parameters Parameters for this facility are:

string	const char *	A OS system command

Return `system()` always returns 0.

See Also "getenv" on page 326

wcstombs

Description Translate a `wchar_t` type character array to a multibyte character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `wcstombs()` function converts a character array containing `wchar_t` type characters to a character array containing multibyte characters. The `wchar_t` type is defined in `stddef.h`.

The Metrowerks C implementation of `wcstombs()` performs no translation; it copies a maximum of `n` bytes from the array pointed to by `pwcs` to the array pointed to by `s`. The function terminates prematurely if a null character is reached.

Return `wcstombs()` returns the number of bytes copied from `pwcs` to `s`.

See Also "Locale specification" on page 87, "mbstowcs" on page 331

wctomb

Description Translate a `wchar_t` type to a multibyte character.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <stdlib.h>
int wctomb(char *s, wchar_t wchar);
```

Parameters Parameters for this facility are:

s	char *	A multibyte string buffer
wchar	wchar_t	A wide character to convert

Remarks The `wctomb()` function converts a `wchar_t` type character to a multibyte character.

The Metrowerks C implementation of `wctomb()` performs no translation; it assigns `wchar` to the character pointed to by `s`.

Return `wctomb()` returns 1 if s is not null and returns 0 otherwise.

See Also "Locale specification" on page 87

"mbtowc" on page 332

27

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

- "strcasecmp" on page 352, string ignore case compare
- "strcat" on page 353 concatenates strings
- "strchr" on page 354 searches by character
- "strcmp" on page 355 compares strings
- "strcpy" on page 358 copies strings
- "strcoll" on page 357 compares string lexicographically
- "strcspn" on page 360 find a substring in a string
- "strdup" on page 361, (Windows _strdup) duplicates strings
- "strerror" on page 362 retrieves and error message from and errno variable
- "strlen" on page 363 returns strings length
- "strpbrk" on page 370 look for an occurrence of a character from one string in another
- "strrchr" on page 371 searches a string for a character

- “_strrev” on page 372, a string reversing function
- “strspn” on page 373 search for a character not in one string in another
- “strstr” on page 374 searches a string for a string
- “strtok” on page 375 retrieves the next token or substring
- “strxfrm” on page 377 transform a string to a locale
- “_strupr” on page 379, string to uppercase string

A function with an `strn` prefix operates on character arrays of a length specified as a function argument. The `strn` functions are:

- “strncasecmp” on page 364, string case compare with length specified
- “strncasecmp” on page 364 string concatenate with length specified
- “strncmp” on page 366 string compare with length specified
- “strncpy” on page 368 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 346 searches a memory block for a character
- “memcmp” on page 349 compares a memory block
- “memcpy” on page 350 copies a memory block
- “memmove” on page 351 moves a memory block
- “memset” on page 352 sets a value for a memory block

A function with a ‘stri’ prefix operates on strings ignoring case.

- “_stricmp” on page 363, string compare ignore case
- “_strnicmp” on page 369, string compare ignore case with length specified

memchr

Description Search for an occurrence of a character.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <string.h>
void *memchr(const void *s, int c, size_t n);
```

Parameters Parameters for this facility are:

s	const void *	The memory to search
c	int	The char to search for
n	size_t	The maximum length to search

Remarks The `memchr()` function looks for the first occurrence of `c` in the first `n` characters of the memory area pointed to by `s`.

Return `memchr()` returns a pointer to the found character, or a null pointer (`NULL`) if `c` cannot be found.

See Also "strchr" on page 354

"strrchr" on page 371

Listing 27.1 Example of memchr() usage.

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

#define ARRAYSIZE 100

int main(void)
{
  // s1 must by same length as s2 for this example!
  static char s1[ARRAYSIZE] = "laugh* giggle 231!";
  static char s2[ARRAYSIZE] = "grunt sigh# snort!";
  char dest[ARRAYSIZE];
  char *strptr;
  int len1, len2, lendest;
```

```
  // Clear destination string using memset()
  memset( (char *)dest, '\0', ARRAYSIZE);

  // 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

Description Compare two blocks of memory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 355

"strncmp" on page 366

Listing 27.2 For example of memcmp() usage

Refer to"Example of memchr() usage." on page 347.

memcpy

Description Copy a contiguous memory block.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 351
"strcpy" on page 358
"strncpy" on page 368

Listing 27.3 For example of memcpy() usage

Refer to "Example of memchr() usage." on page 347.

memmove

Description Copy an overlapping contiguous memory block.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 350
"memset" on page 352
"strcpy" on page 358
"strncpy" on page 368

Listing 27.4 For example of memmove() usage

Refer to "Example of memchr() usage." on page 347.

memset

Description Clear the contents of a block of memory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 27.5 For example of memset() usage

Refer to "Example of memchr() usage." on page 347 .

strcasecmp

Description Ignore case string comparison function

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Return Strcasecmp 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 "strncasecmp" on page 364

strcat

Description Concatenate two character arrays.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 "strncasecmp" on page 364

Listing 27.6 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

Description Search for an occurrence of a character.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `strchr()` function searches for the first occurrence of the character `c` in the character array pointed to by `s`. The `s` argument must point to a null terminated character array.

Return `strchr()` returns a pointer to the successfully located character. If it fails, `strchr()` returns a null pointer (`NULL`).

See Also "memchr" on page 346

"strrchr" on page 371

Listing 27.7 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

Description Compare two character arrays.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <string.h>
int strcmp(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 `strcmp()` function compares the character array pointed to by `s1` to the character array pointed to by `s2`. Both `s1` and `s2` must point to null terminated character arrays.

Return `strcmp()` returns a zero if `s1` and `s2` are equal, a negative value if `s1` is less than `s2`, and a positive value if `s1` is greater than `s2`.

See Also "memcmp" on page 349
"strcoll" on page 357
"strncmp" on page 366

Listing 27.8 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

Description Compare two character arrays according to locale.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `strcoll()` function compares two character arrays based on the collating sequence set by the `locale.h` header file.

The Metrowerks C implementation of `strcoll()` compares two character arrays using `strcmp()`. It is included in the string library to conform to the ANSI C Standard Library specification.

Return `strcoll()` returns zero if `s1` is equal to `s2`, a negative value if `s1` is less than `s2`, and a positive value if `s1` is greater than `s2`.

See Also "Locale specification" on page 87

"memcmp" on page 349

"strcmp" on page 355

"strncmp" on page 366,

Listing 27.9 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 < 1)
    printf("%s is less than %s\n", s1, s2);
  else
    printf("%s is equal or greater than %s\n", s1, s2);

  return 0;
}
```

```
Output:
aardvark is less than xylophone
```

strcpy

Description Copy one character array to another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 350
"memmove" on page 351
"strncpy" on page 368

Listing 27.10 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

Description Count characters in one character array that are not in another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 counts the initial length of the character array pointed to by `s1` that does not contain characters in the character array pointed to by `s2`. The function starts counting characters at the beginning of `s1` and continues counting until a character in `s2` matches a character in `s1`.

Both `s1` and `s2` must point to null terminated character arrays.

Return `strcspn()` returns the length of characters in `s1` that does not match any characters in `s2`.

See Also "strpbrk" on page 370

"strspn" on page 373

Listing 27.11 Example of strcspn() usage.

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

int main(void)
{
  static char s1[] = "chocolate *cinnamon* 2 ginger";
  static char s2[] = "1234*";
```

```
  printf(" s1 = %s\n s2 = %s\n", s1, s2);
  printf(" %d\n", strcspn(s1, s2));

  return 0;
}
```

```
Output:
 s1 = chocolate *cinnamon* 2 ginger
 s2 = 1234*
 10
```

strdup

Description Duplicate a string.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <string.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

NOTE: This function is defined in extras.c but not included in the standard library or headers for other than Windows systems.

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

Remarks The Windows routines use a leading underscore.

See Also "memcpy" on page 350

strerror

Description Return an error message in a character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <string.h>
char *strerror(int errnum);
```

Parameters Parameters for this facility are:

errnum	int	Provides an index of errno

Remarks The `strerror()` function returns a pointer to a null terminated character array that contains an error message. The `errnum` argument is returned by `strerror();` in a string.

Return `strerror()` returns a pointer to a null terminated character array containing an error message.

Listing 27.12 Example of strerror() usage.

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

int main(void)
{
  puts(strerror(8));

  return 0;
}
```

```
Output:
unknown error (8)
```

_stricmp

Description A function for string comparison ignoring case.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <string.h>
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 str 1. If they are equal returns zero.

See Also "strcmp" on page 355

" strnicmp" on page 369

strlen

Description Compute the length of a character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 27.13 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.
```

strncasecmp

Description Ignore case string comparison function with length specified.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this function are:

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

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 352

strncat

Description Append a specified number of characters to a character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 353

Listing 27.14 Example of strncat() usage.

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

int main(void)
{
  static char s1[100] = "abcdefghijklmnopqrstuv";
  static char s2[] = "wxyz0123456789";

  strncat(s1, s2, 4);
  puts(s1);

  return 0;
}
```

```
Output:
abcdefghijklmnopqrstuvwxyz
```

strncmp

Description Compare a specified number of characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters Parameters for this facility are:

s1	const char *	The string to compare
s2	const char *	The comparison string
n	size_t	The maximum length 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`. Both `s1` and `s2` do not necessarily have 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 349

"strcmp" on page 355

Listing 27.15 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

Description Copy a specified number of characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `strncpy()` function copies a maximum of `n` characters from the character array pointed to by `source` to the character array pointed to by `dest`. Neither `dest` nor `source` must necessarily point to null terminated character arrays. Also, `dest` and `source` must not overlap.

If a null character (`'\0'`) is reached in `source` before `n` characters have been copied, `strncpy()` continues padding `dest` with null characters until `n` characters have been added to `dest`.

The function does not terminate `dest` with a null character if `n` characters are copied from `source` before reaching a null character.

Return `strncpy()` returns the value of `dest`.

See Also "memcpy" on page 350

"memmove" on page 351

"strcpy" on page 358

Listing 27.16 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
```

_strnicmp

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

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

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

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 Strnicmp 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 355
" stricmp" on page 363

strpbrk

Description Look for the first occurrence of an array of characters in another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 360

Listing 27.17 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

Description Search for the last occurrence of a character.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 346

"strchr" on page 354

Listing 27.18 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
```

_strrev

Description Strrev is a function that reverses a string.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

```
#include <string.h>
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 358

strspn

Description Count characters in one character array that are in another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 count
s2	const char *	A list of characters to look for

Remarks The `strspn()` function counts the initial number of characters in the character array pointed to by `s1` that contains characters in the character array pointed to by `s2`. The function starts counting characters at the beginning of `s1` and continues counting until it finds a character that is not in `s2`.

Both `s1` and `s2` must point to null terminated character arrays.

Return `strspn()` returns the number of characters in `s1` that matches the characters in `s2`.

See Also "strpbrk" on page 370

"strcspn" on page 360

Listing 27.19 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

Description Search for a character array within another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 346

"strchr" on page 354

Listing 27.20 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

Description Extract tokens within a character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 tokenizes the character array pointed to by `str`. The `sep` argument points to a character array containing token separator characters. The tokens in `str` are extracted by successive calls to `strtok()`.

The first call to `strtok()` causes it to search for the first character in `str` that does not occur in `sep`. The function returns a pointer to the beginning of this first token. If no such character can be found, `strtok()` returns a null pointer (`NULL`).

If, on the first call, `strtok()` finds a token, it searches for the next token.

The function searches by skipping characters in the token in `str` until a character in `sep` is found. This character is overwritten with a null character to terminate the token string, thereby modifying the character array contents. The function also keeps its own pointer to the character after the null character for the next token. Subsequent token searches continue in the same manner from the internal pointer.

Subsequent calls to `strtok()` with a NULL `str` argument cause it to return pointers to subsequent tokens in the original `str` character array. If no tokens exist, `strtok()` returns a null pointer. The `sep` argument can be different for each call to `strtok()`.

Both `str` and `sep` must be null terminated character arrays.

Return When first called `strtok()` returns a pointer to the first token in `str` or returns a null pointer if no token can be found.

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.

`strtok()` modifies the character array pointed to by `str`.

Listing 27.21 Example of strtok() usage.

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

int main(void)
{
  static char s[50] = "(ape+bear)*(cat+dog)";
  char *nexttok;

  // first call to strtok()
  puts(strtok(s, "()+*"));

  nexttok = strtok(NULL, "()+*");
  puts(nexttok);

  nexttok = strtok(NULL, "()+*");
  puts(nexttok);

  nexttok = strtok(NULL, "()+*");
  puts(nexttok);

  return 0;
}
```

```
Output:
ape
bear
cat
dog
```

strxfrm

Description Transform a locale-specific character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 to conform to the locale character set defined in `locale.h`.

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 87

"strcpy" on page 358

Listing 27.22 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
```

_strupr

Description Strupr converts a string to uppercase.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	**Win32**	

Prototype

```
char *_strupr(char *str);
```

Parameters Parameters for this function are:

str	char	The string being converted

Return A pointer to the reversed string.

See Also "toupper" on page 54
"tolower" on page 53

28

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 382 is a structure for storing time data.
- "tzname" on page 383, an array that stores the time zone abbreviations
- "asctime" on page 384 to convert a tm structure type to a char array
- "clock" on page 385 to determine the time since the computer was started
- "ctime" on page 386 to convert a time_t type to a char array
- "difftime" on page 387 to determine the difference between two times
- "gmtime" on page 388 to determine Greenwich Mean Time
- "localtime" on page 389 to determine the local time
- "mktime" on page 390 to convert a tm structure to time_t type
- "strftime" on page 392 to format time as a C string
- "_strdate" on page 391, stores a date in a string buffer
- "time" on page 397 to determine a number of seconds from a set time
- "tzset" on page 398, 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`.

The `clock_t` type is a numeric, system dependent type returned by the `clock()` function.

The `time_t` type is a system dependent type used to represent a calendar date and time.

NOTE: The ANSI/ISO C Standard does not specify a start date, therefore an arbitrarily chosen Jan. 1, 1900 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

Description The `struct tm` type contains a field for each part of a calendar date and time.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <time.h>
struct tm {
   int tm_sec;
   int tm_min;
   int tm_hour;
   int tm_mday;
   int tm_mon;
   int tm_year;
   int tm_wday;
   int tm_yday;
```

```
int tm_isdst;
};
```

Remarks The `tm` structure members are listed "Tm Structure Members." on page 383.

NOTE: The `tm_isdst` flag is positive if Daylight Savings Time is in effect, zero if it is not, and negative if such information is not available.

Table 28.1 Tm Structure Members.

Field	Description	Range min - max
`int tm_sec`	Seconds after the minute	0 - 59
`int tm_min`	Minutes after the hour	0 - 59
`int tm_hour`	Hours after midnight	0 - 23
`int tm_mday`	Day of the month	1 - 31
`int tm_mon`	Months after January	0 - 11
`int tm_year`	Years after 1900	
`int tm_wday`	Days after Sunday	0 - 6
`int tm_yday`	Days after January 1	0 - 365
`int tm_isdst`	Daylight Savings Time flag	

tzname

Description The _tzname_ array contains the names (abbreviations) of the time zones for local standard time and DST.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <time.h>
extern char *tzname[2];
```

See Also "tzset" on page 398

asctime

Description Convert a `tm` structure to a character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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, pointe d 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 %d %H:%M: %S %Y"`.

Return `asctime()` returns a null terminated character array pointer containing the converted `tm structure`.

See Also "ctime" on page 386

"strftime" on page 392

Listing 28.1 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

Description Return the amount of time the system has been running.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

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

Parameters None

Remarks The `clock()` function returns the amount of time since the computer system was started. To compute the time in seconds, divide the `clock_t` value by `CLOCKS_PER_SEC`, a macro defined in `time.h`.

Return `clock()` returns a `clock_t` type value representing the time since the system was started.

Listing 28.2 Example of clock() usage.

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

```
int main(void)
{
  clock_t uptime;

  uptime = clock() / CLOCKS_PER_SEC;

  printf("I was booted %ul seconds ago.\n", uptime);

  return 0;
}
```

```
Output:
I was booted 24541 seconds ago.
```

ctime

Description Convert a `time_t` type to a character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 used by `asctime()`.

Return `ctime()` returns a null terminated character array pointer containing the converted `time_t` type.

See Also "asctime" on page 384
"strftime" on page 392

Listing 28.3 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

Description Compute the difference between two `time_t` types.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 28.4 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

Description Convert a `time_t` value to Coordinated Universal Time (UTC), which is the new name for Greenwich Mean Time.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 28.5 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

Description Convert a `time_t` type to a `struct tm` type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 390

For Usage Refer to the example for "Example of difftime usage." on page 388.

mktime

Description Convert a `struct tm` item to a `time_t` type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 389

Listing 28.6 For example of usage

```
Refer to the example for "Example of difftime usage." on page 388.
```

_strdate

Description The strdate function stores a date in a buffer provided.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <time.h>
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 392

strftime

Description Format a `tm` structure.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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:

s	char *	The string to format
format	const char *	The format string
timeptr	const struct tm*	The address of the time structure

Remarks The `strftime()` function converts a `tm` structure to a character array using a programmer supplied format.

The `s` argument is a pointer to the array to hold the formatted time.

The `maxsize` argument specifies the maximum length of the formatted character array.

The `timeptr` argument points to a `tm` structure containing the calendar time to convert and format.

The format argument points to a character array containing normal text and format specifications similar to a `printf()` function format string. Format specifiers are prefixed with a percent sign (%). Doubling the percent sign (%%) will output a single %.

NOTE: Refer to "strftime() conversion characters" on page 393 for a list of format specifiers.

Table 28.2 strftime() conversion characters

Char	Description
a	Abbreviated weekday name.
A	Full weekday name.
b	Abbreviated month name.
B	Full month name.
c	The strftime() format equaling the format string of `"%x %X"`.
d	Day of the month as a decimal number.
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.
p	`"AM"` or `"PM"`.
S	The second as a decimal number from 00 to 59.
U	The week number of the year as a decimal number from 00 to 52. Sunday is considered the first day of the week.
w	The weekday as a decimal number from 0 to 6. Sunday is (0) zero.
W	The week of the year as a decimal number from 00 to 51. Monday is the first day of the week.

Char	Description
x	The date representation of the current locale.
X	The time representation of the current locale.
y	The last two digits of the year as a decimal number.
Y	The century as a decimal number.
z	The time zone name or nothing if it 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 28.7 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);
 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 Thu
```

```
Today's full name is Thursday
Today's aabr.month name is Aug
Today's full month name is August
Today's date and time is Aug 24 11:42:16 1995
The day of the month is 24
The 24-hour clock hour is 11
The 12-hour clock hour is 11
Today's day number is 236
Today's month number is 08
The minute is 42
The AM/PM is AM
The second is 16
The week number of the year,starting on a Sunday is 34
The number of the week is 4
The week number of the year,starting on a Monday is 34
The date is Aug 24 1995
The time is 11:42:16
The last two digits of the year are 95
The year is 1995
The time zone cannot be determined
```

time

Description Return the current system calendar time.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 28.8 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

Description 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 is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 383

29

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 402 change the directory
- "close" on page 404 close a file opened with open
- "cuserid" on page 407 retrieves the current user's ID
- "exec" on page 409 executes programs from within a program
- "getcwd" on page 411 gets the current working directory
- "getlogin" on page 412 returns a login name
- "getpid" on page 413 returns the process ID
- "isatty" on page 414 determines if a file ID is attached to a terminal
- "lseek" on page 416 seek when opened with open
- "read" on page 417 read when opened with open
- "rmdir" on page 418 removes a directory or folder
- "sleep" on page 419 pauses a program
- "ttyname" on page 420 determines a terminal id

- "unlink" on page 421 deletes a file

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.

chdir

Description Change the current directory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unistd.h>
int chdir(const char *path);
```

Parameters Parameters for this facility are:

path	char *	The new pathname

Remarks The function `chdir()` is used to change from one directory to a different directory or folder. Example of usage is given in "Example of chdir() usage." on page 402

Return `chdir()` returns zero, if successful. If unsuccessful `chdir()` returns negative one and sets `errno`.

See Also "Overview of errno.h" on page 59

Listing 29.1 Example of chdir() usage.

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

```
#include <unistd.h>
#include <stat.h>

#define SIZE FILENAME_MAX
#define READ_OR_WRITE0x0 /* 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
```

close

Description Close an open file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unistd.h>
int close(int fildes);
```

Parameters Parameters for this facility are:

fildes	int	The file descriptor

Remarks The `close()` function closes the file specified by the argument. This argument is the value returned by `open()`. Example of usage is given in "Example of close() usage." on page 405

Return If successful, `close()` returns zero. If unsuccessful, `close()` returns negative one and sets `errno`.

See Also "open" on page 67
"fclose" on page 221
"errno" on page 59

Listing 29.2 Example of close() usage.

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


#define SIZE FILENAME_MAX
#define MAX 1024

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


int main(void)
{
```

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

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

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

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

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

Description Retrieve the current user's ID.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 408

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 29.3 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
```

exec

Description Load and execute a child process within the current program memory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Remark On the Macintosh, all exec family calls pass through `exec()`, because argument passing (`argc`, `argv`) doesn't exist for Mac application.

Prototype

```
#include <unistd.h>

int exec(const char *path, ...);
int execl(const char *path, ...);
int execle(const char *path, ...);
int execlp(const char *path, ...);
int execv(const char *path, ...);
intexecve(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:

path	const char *	The commandline pathname to execute
...		A variable list of arguments

Table 29.1 The exec() type functions

UNIX Function	On the Macintosh System
`#define execl`	`exec`
`#define execv`	`exec`
`#define execle`	`exec`
`#define execve`	`exec`
`#define execlp`	`exec`
`#define execvp`	`exec`

Description Launches the application named and then quits upon successful launch. Example of usage is given in "Example of exec() usage." on page 410.

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 and WinSIOUX" on page 179
"Overview of errno.h" on page 59

Listing 29.4 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

Description Get the current directory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unistd.h>
char *getcwd(char *buf, int size);
```

Parameters Parameters for this facility are:

buf	char	A buffer to hold 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 59

Listing 29.5 For example of getcwd usage

Refer to "Example of chdir() usage." on page 402.

getlogin

Description Retrieve the username that started the process.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 412

NOTE: The Mac doesn't have a login, so this function returns the Owner Name from the Sharing Setup Control Panel

Return `getlogin()` returns a character pointer.

Listing 29.6 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

Description Retrieve the process identification number.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unistd.h>
```

Table 29.2 getpid() Macros

Macro	Represents
`#define getpid()`	Process ID
`#define getppid()`	Parent process ID
`#define getuid()`	Real user ID
`#define geteuid()`	Effective user ID
`#define getgid()`	Real group ID
`#define getegid()`	Effective group ID
`#define getpgrp()`	Process group ID

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 414

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 29.7 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

Description Determine a specified file_id

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unistd.h>
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() ttyname() usage." on page 415

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 30

Listing 29.8 Example of isatty() ttyname() 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",
        ttyname(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

Description Seek a position on a file stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unistd.h>
long lseek(int fildes, long offset, int origin);
```

Parameters Parameters for this facility are:

fildes	int	The file descriptor
offset	long	The offset to move in bytes
origin	int	The starting point of the seek

Remarks The function `lseek()` sets the file position location a specified byte offset from a specified initial location.

The origin of the offset must be one of the positions in "The lseek offset positions." on page 417.

Table 29.3 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 number of bytes offset. If unsuccessful, it returns a value of negative one long integer.

See Also "fseek" on page 256
"ftell" on page 260
"open" on page 67

Listing 29.9 For example of lseek() usage

Refer to "Example of close() usage." on page 405.

read

Description Read from a file stream that has been opened for unformatted Input/Output.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `read()` copies the number of bytes specified by the count argument, from the file to the character buffer. File reading begins at the current position. The position moves to the end of the read position when the operation is completed.

NOTE: This function should be used in conjunction with `unistd.h:write()`, and `fcntl.h:open()` only.

Return `read()` returns the number of bytes actually read from the file. In case of an error a value of negative one is returned and errno is set.

See Also "fread" on page 248

"open" on page 67

Listing 29.10 For example of read() usage

Refer to "Example of close() usage." on page 405.

rmdir

Description Delete a directory or folder.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unistd.h>
int rmdir(const char *path);
```

Parameters Parameters for this facility are:

path	const char *	The pathname of the directory being removed

Remarks The `rmdir()` function removes the directory (folder) specified by the argument.

Return If successful, `rmdir()` returns zero. If unsuccessful, `rmdir()` returns negative one and sets `errno`.

See Also "mkdir" on page 202

"errno" on page 59

Listing 29.11 For example of rmdir() usage

```
Refer to"Example of chdir() usage." on page 402.
```

sleep

Description Delay program execution for a specified number of seconds.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `sleep()` delays execution of a program for the time indicated by the unsigned integer argument. For the Macintosh system there is no error value returned. Example of usage is given in "Example of sleep() usage." on page 420

Return `sleep()` returns zero.

Listing 29.12 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
```

ttyname

Description Retrieve the name of the terminal associated with a file ID.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 29.13 For example of ttyname() usage

```
Refer to "Example of isatty() ttyname() usage." on page 415.
```

unlink

Description Delete (unlink) a file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `unlink()` removes the specified file from the directory. Example of usage is given in "Example of unlink() usage." on page 422

Return If successful, `unlink()` returns zero. If unsuccessful, it returns a negative one.

See Also "rmdir" on page 418

"mkdir" on page 202

Listing 29.14 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

Description Write to an un-formatted file stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unistd.h>
int write(int fildes, const char *buf, int count);
```

Parameters Parameters for this facility are:

fildes	int	The file descriptor
buf	const char *	The addrss of the buffer being written
count	int	The size of the buffer being written

Remarks The function `write()` copies the number of bytes in the `count` argument from the character array buffer to the file `fildes`. The file position is then incremented by the number of bytes written.

NOTE: This function should be used in conjunction with "read" on page 417, and "open" on page 67 only.

Return `write()` returns the number of bytes actually written.

See Also "fwrite" on page 261

"read" on page 417

"open" on page 67

Listing 29.15 For example of write() usage

```
Refer to "Example of close() usage." on page 405.
```

30

unix.h

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

Overview of unix.h

The header file `unix.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 globals and facilities in unix.h are:

- "_fcreator" on page 426 sets a Macintosh file creator
- "_ftype" on page 426 sets a Macintosh file type
- "fdopen" on page 428 converts a file descriptor to a stream
- "fileno" on page 429 converts a stream to a file descriptor
- "tell" on page 430 determines the offset of a file

unix.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.

Globals

Description Global variables for setting the type and creator of new files

_fcreator

Description To specify a Macintosh file creator.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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 427 is an example of the use of the global variable _fcreator.

_ftype

Description To specify a Macintosh file type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	**Mac OS**	Palm OS	Win32	

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 427 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 30.1 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
```

fdopen

Description Converts a file descriptor to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <unix.h>
FILE *fdopen(int fildes, char *mode);
```

Parameters Parameters for this facility are:

fildes	int	The file descriptor
mode	char *	The file open 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 429

"open" on page 67

Listing 30.2 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;
}
```

fileno

Description Converts a stream to a file descriptor

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 creates a 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 30.1 File Descriptors for the Standard I/O Streams

This function call…	Returns this file descriptor…
`fileno(stdin)`	`0`
`fileno(stdout)`	`1`
`fileno(stderr)`	`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 428
"open" on page 67

Figure 30.1 Example of fdopen() usage.

```
#include <unix.h>

int main(void)
{
  write(fileno(stdout), "Hello world!\n", 13);

  return 0;
}
```

```
Reult
Access time:       Tue Apr 18 22:28:22 1995
Modification time: Tue Apr 18 22:28:22 1995
Creation time:     Tue Apr 18 11:28:41 1995
Block size:        11264
Number of blocks:  1
```

tell

Description Returns the current offset for a file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 260

"lseek" on page 416

Listing 30.3 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.
```

31

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 433 to set a file modification time
- "utimes" on page 436 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.

utime

Purpose Sets a file's modification time.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <utime.h>
int utime(const char *path,
      const struct utimbuf *buf);
```

Parameters

Parameters for this facility are:

path	const char *	The pathname as a string
buf	const struct utimbuf *	The address of a stuct that will hold a file's time information

Remarks

This function sets the modification time for the file specified in `path`. Since the Macintosh does not have anything that corresponds to a file's access time, it ignores the `actime` field in the `utimbuf` structure.

If `buf` is `NULL`, `utime()` sets the modification time to the current time. If `buf` points to a `utimbuf` structure, `utime()` sets the modification time to the time specified in the `modtime` field of the structure.

The `utimbuf` structures contains the fields in Table 31.2.

Table 31.1 The utimbuf structure

This field…		is the…
time_t	actime	Access time for the file. Since the Macintosh has nothing that corresponds to this, `utime()` ignores this field.
time_t	modtime	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 436

"ctime" on page 386
"mktime" on page 390
"stat" on page 204
"fstat" on page 201

Listing 31.1 Example for utime()

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

int main(void)
{
  struct utimbuf timebuf;
  struct tm date;
  struct stat info;


  /* Create a calendar time for
  Midnight, Apr. 4, 1964. */
  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=64; /* ...in 1964     */
  date.tm_isdst=-1;/* Not daylight savings */
  timebuf.modtime=mktime(&date);
    /* Convert to calendar time.    */

  /* Change modification date to  *
   * midnight, Apr. 4, 1964.   */
  utime("mytest", &timebuf);
  stat("mytest", &info);
  printf("Mod date is %s", ctime(&info.st_mtime));

  /* Change modification date to */
   * right now.                  */
  utime("mytest", NULL);
  stat("mytest", &info);
```

```
  printf("Mod date is %s", ctime(&info.st_mtime));

  return 0;
}
```

```
This program might print the following to standard output:
Mod date is Sat Apr  4 00:00:00 1964
Mod date is Mon Apr 10 20:43:09 1995
```

utimes

Description Sets a file's modification time

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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	timeva 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 31.2.

Table 31.2 The timeval structure

This field		is the
int t	tv_sec	Seconds
int	tv_usec	Microseconds. Since the Macintosh does not use microseconds, `utimes()` ignores this.

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 433
"ctime" on page 386
"mktime" on page 390
"fstat" on page 201
"stat" on page 204

Listing 31.2 Example for utimes()

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

int main(void)
{
  struct tm date;
  struct timeval buf[2];
  struct stat info;


/* Create a calendar time for
```

```
Midnight, Sept. 9, 1965.*/
  date.tm_sec=0;      /* Zero seconds */
  date.tm_min=0;      /* Zero minutes */
  date.tm_hour=0;     /* Zero hours */
  date.tm_mday=9;     /* 9th day */
  date.tm_mon=8;      /* .. of September  */
  date.tm_year=65;    /* ...in 1965   */
  date.tm_isdst=-1;   /* Not daylight savings  */
  buf[1].tv_sec=mktime(&date);
  /* Convert to calendar time.  */

  /* Change modification date to   *
   * midnight, Sept. 9, 1965.      */
  utimes("mytest", buf);
  stat("mytest", &info);
  printf("Mod date is %s", ctime(&info.st_mtime));

  return 0;
}
```

```
This program prints the following to standard output:
Mod date is Thu Sep  9 00:00:00 1965
```

32

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 439 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.

uname

Description Gets information about the system you are using.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 32.1. All the fields are null-terminated strings.

Table 32.1 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 201

"stat" on page 204

Listing 32.1 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:        Chan's PowerMac
Release:          7
Version:          51
Machine:            Power Macintosh
This machine is a Power Macintosh running Version 7.5.1 of the
MacOS. The Macintosh Name field of the Sharing Setup control panel
contains "Chan's PowerMac"
```

33

wchar.h

The header file.

Overview of wchar.h

The header file wchar.h contains defines and functions to manipulate wide character sets.

The header `wchar.h` has many diverse functions and definitions.

Input and output facilities

- "fgetwc" behaves like `fgetc` for wide characters
- "fgetws" behaves like `fgets` for wide characters
- "fputwc" behaves like `fputc` for wide characters
- "fputws" behaves like `fputs` for wide characters
- "fwprintf" behaves like `fprintf` for wide characters
- "fwscanf" behaves like `fscanf` for wide characters
- "getwc" behaves like `getc` for wide characters
- "getwchar" behaves like `getchar` for wide characters
- "putwc" behaves like `putc` for wide characters
- "putwchar" behaves like `putchar` for wide characters
- "swprintf" behaves like `sprintf` for wide characters
- "swscanf" behaves like `sscanf` for wide characters
- "wprintf" behaves like `printf` for wide characters
- "wscanf" behaves like `scanf` for wide characters
- "vfwprintf" behaves like `vfprintf` for wide characters
- "__vfwscanf" behaves like `vfscanf` for wide characters
- "__vswscanf" behaves like `vsscanf` for wide characters

- "vwprintf" behaves like `vprintf` for wide characters
- "vswprintf" behaves like `fgetc` vsprintf for wide characters
- "vwscanf" behaves like `vscanf` for wide characters

Time facilities

- "wasctime" behaves like `asctime` for wide characters
- "wcsftime" behaves like `csftime` for wide characters
- "wctime" behaves like `ctime` for wide characters

Mapping facilities

- "towctrans" a case and wide character mapping function
- "wctrans" a wide character mapping function

String facilities

- "watof" behaves like `atof` for wide characters
- "wcscat" behaves like `strcat` for wide characters
- "wcschr" behaves like `strchr` for wide characters
- "wcscmp" behaves like `strcmp` for wide characters
- "wcscspn" behaves like `strspn` for wide characters
- "wcscoll" behaves like `strcoll` for wide characters
- "wcscpy" behaves like `strcpy` for wide characters
- "wcslen" behaves like strlen for wide characters
- "wcsncat" behaves like `strncat` for wide characters
- "wcsncmp" behaves like `strncmp` for wide characters
- "wcsncpy" behaves like `strncpy` for wide characters
- "wcspbrk" behaves like `strbrk` for wide characters
- "wcstod" behaves like `strtod` for wide characters
- "wcsrchr" behaves like `strrchr` for wide characters
- "wcsspn" behaves like `strspn` for wide characters
- "wcsstr" behaves like `strstr` for wide characters
- "wcstok" behaves like `strtok` for wide characters
- "wcsxfrm" behaves like `strxfrm` for wide characters

- "wmemcpy" behaves like `memcpy` for wide characters
- "wmemmove" behaves like `memmove` for wide characters
- "wmemset" behaves like `memset` for wide characters
- "wmemchr" behaves like `memchr` for wide characters
- "wmemcmp" behaves like `memcmp` for wide characters

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 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.

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 33.1 Wide Character Definitions

Defines	Definitions
WCHAR_MIN	Minimum size of a wide char
WCHAR_MAX	Maximum size of a wide char
WEOF	End of file
EILSEQ	Wide char sequence error
wctrans_t	Scalar type for locale specific wide char mappings

fgetwc

Description Gets a wide character from a file stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t  fgetwc(FILE * file);
```

Parameters Parameters for this function are:

file	FILE *	The file to retrieve from

Remarks Performs the same task as `fgetc` for wide character

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return Returns the character or `WEOF` for an error

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"fgetc" on page 229

fgetws

Description The function fgetws reads a wide character string from a file stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t *fgetws(wchar_t * s, int n, FILE * file);
```

Parameters Parameters for this function are:

s	wchar_t *	A wide char string for input
n	int	The number of wide char
file	FILE *	A pointer to the file stream

Remarks Behaves like `fgets` for wide characters.

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return Returns a pointer to `'s'` if successful or `FEOF` or `NULL` for a failure.

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"fgets" on page 233

fwprintf

Description Formatted file insertion

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
int   fwprintf(FILE * file,
   const wchar_t * format, ...);
```

Parameters Parameters for this function are:

file	FILE *	A pointer to the 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 `stream` and advances the file position indicator. Its operation is the same as w`printf()` with the addition of the `stream` argument. Refer to the description of `printf()`

NOTE: 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 445

"fprintf" on page 238

fputwc

Description Inserts a single wide character into a file stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 file stream

Remarks Performs the same task as `fputc` for a wide character type.

NOTE: 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 445

"fputc" on page 245

fputws

Description Inserts a wide character array into a file stream

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 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()).

NOTE: 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 445

"fputs" on page 246

fwscanf

Description Reads 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 <wchar.h>
int fwscanf(FILE * file,
   const wchar_t * format, ...);
```

Parameters Parameters for this function are:

file	FILE *	The 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 `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 `scanf()` function description.

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return Returns the number of items read. If there is an error in reading data that is inconsistent with the format string, `fwscanf()` sets `errno` to a nonzero value. `fwscanf()` returns `WEOF` if it reaches the end-of-file.

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"fscanf" on page 252

getwc

Description Returns a wide character type from a file stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 445

"getc" on page 262

getwchar

Description Returns a wide character type from the standard input.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t  getwchar(void);
```

Parameters Has no parameters.

Remarks Performs the same task as `getchar` for a wide character type.

Return Returns the value of the next character from `stdin` as an `int` if it is successful. `getwchar()` returns `WEOF` if it reaches an end-of-file or an error occurs.

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"getwchar" on page 452

putwc

Description Write a wide character type to a stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t  putwc(wchar_t c, FILE * file);
```

Parameters Parameters for this function are:

c	wchar_t	The value to compute
file	FILE	The output stream

Remarks Performs the same task as `putc` for a wide character type.

Return Returns the character written when successful and returns `WEOF` when it fails

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"putc" on page 275

putwchar

Description Writes a character to standard output.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 445

"putchar" on page 277

swprintf

Description Formats text in a wide character type string.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 number of wide characters permissible 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.

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 445

"fwprintf" on page 447

"sprintf" on page 292

swscanf

Description Reads a formatted wide character string.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `sscanf` for a wide character type.

Return Returns the number of items successfully read and converted and returns `WEOF` 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 445

"sscanf" on page 293

towctrans

Description Maps a wide character type to another wide character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 475

__vfwscanf

Description A variable argument for reading a formatted file.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `fscanf` for a wide character type.

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Listing 0.2 Return

Returns the number of items scanned if successful.

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"fscanf" on page 252

__vswscanf

Description A variable argument for reading a formatted string.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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
....		Variable arguments

Remarks Performs the same task as `sscanf` for a wide character type.

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return Returns the number of items scanned if successful.

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"sscanf" on page 293

vwscanf

Description A variable argument for reading a formatted string.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 Performs the same task as `sscanf` for a wide character type.

Return Returns the number of items scanned if successful.

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"__vfwscanf" on page 456

"scanf" on page 284

vfwprintf

Description Write a formatted text to a file stream.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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
....		Variable arguments

Remarks Performs the same task as `vfpirntf` for a wide character type.

NOTE: On embedded/ RTOS systems this function only is implemented for stdin, stdout and stderr files.

Return Returns the number of characters written or `WEOF` if it failed.

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"vfprintf" on page 300

vswprintf

Description Write formatted output to `a string.`

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
int vswprintf(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
....		Variable arguments

Remarks Performs the same task as `vsprintf` for a wide character type.

Return Returns the number of characters written or `WEOF` if it failed..

See Also "Wide Character and Byte Character Stream Orientation" on page 445

"vsprintf" on page 304

vwprintf

Description Write a variable argument formatted output to `stdout`.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
int vwprintf(const wchar_t * format, va_list arg);
```

Parameters Parameters for this function are:

format	wchar_t*	The format string
....		Variable arguments

Remarks Performs the same task as `vprintf` for a wide character type.

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 445

"vprintf" on page 302

wasctime

Description Convert a `tm` structure to a wide character array

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t * wasctime(const struct tm * tm);
```

Parameters Parameters for this function are:

tm	const struct tm *	A structure to hold a time value

Remarks Performs the same task as `asctime` for a wide character type.

Return Returns a null terminated character array pointer containing the converted `tm structure`.

See Also "asctime" on page 384

watof

Description Convert a wide character string to a double type

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
double watof(wchar_t * str);
```

Parameters Parameters for this function are:

x	double	The value to compute

Remarks Performs the same task as `atof` for a wide character type.

Return Returns the converted value.

See Also "atof" on page 313

wcscat

Description Wide character string concatenation

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 353

wcschr

Description Search for an occurrence of a wide character.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 354

wcscmp

Description Compare two wide character arrays.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
int  cscmp(const wchar_t * str1,
   const wchar_t * str2);
```

Parameters Parameters for this function are:

str1t	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 355
"wmemcmp" on page 476

wcscoll

Description Compare two wide character arrays according to locale.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 357
"wcscmp" on page 463
"wmemcmp" on page 476

wcscspn

Description Counts the number of wide characters in one wide character array that are not in another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 string to find

Remarks Performs the same task as `strcspn` for a wide character type.

Return Returns the length of characters in `str` that does not match any characters in `set`.

See Also "strcspn" on page 360

wcscpy

Description Copy one wide character array to another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t * (wcscpy)(wchar_t * dst,
   const wchar_t * src);
```

Parameters Parameters for this function are:

dst	wchar_t *	The destination string
src	wchar_t *	The source being copied

Remarks Performs the same task as `strcpy` for a wide character type.

Return Returns a pointer to the destination string.

See Also "strcpy" on page 358

wcslen

Description Compute the length of a wide character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 Performs the same task as `strlen` for a wide character type.

Return Returns the number of characters in a character array not including the terminating null character.

See Also "strlen" on page 363

wcsncat

Description Append a specified number of characters to a wide character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t * wcsncat(wchar_t * dst,
   const wchar_t * src, size_t n);
```

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 Performs the same task as `strncat` for a wide character type.

Return Returns a pointer to the destination string.

See Also "strncat" on page 365

wcsncmp

Description Compare a specified number of wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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	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 "strncmp" on page 366

wcsncpy

Description Copy a specified number of wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `strncpy` for a wide character type.

Return Returns a pointer to the destination string.

See Also "strncpy" on page 368
"wcscpy" on page 465

wcspbrk

Description Look for the first occurrence of an array of wide characters in another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t * wcspbrk(const wchar_t * str,
   const wchar_t * set);
```

Parameters Parameters for this function are:

str	wchar_t*	The string being searched
set	wchar_t *	The search string

Remarks Performs the same task as `strpbrk` for a wide character type.

Return Returns a pointer to the first character in `str` that matches any character in `set`, and returns a null pointer (`NULL`) if no match was found.

See Also "strpbrk" on page 370

wcsspn

Description Count the number of wide characters in one wide character array that are in another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
size_t wcsspn(const wchar_t * str,
   const wchar_t * set);
```

Parameters Parameters for this function are:

str	wchar_t *	The searched string
set	const wchar_t *	The string to search for

Remarks Performs the same task as `strspn` for a wide character type.

Return Returns the number of characters in `str` that matches the characters in `set`.

See Also "strspn" on page 373

wcsrchr

Description Search for the last occurrence of a wide character.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `strrchr` 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 371

wcsstr

Description Search for a wide character array within another.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t * wcsstr(const wchar_t * str,
   const wchar_t * pat);
```

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 `strstr` for a wide character type.

Return Returns a pointer to the first occurrence of `s2` in `s1` and returns a null pointer (`NULL`) if `s2` cannot be found.

See Also "strstr" on page 374

"wcschr" on page 463

wcstod

Description Wide character array to numeric conversions.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 Performs the same task as `strtod` for a wide character type.

Return Returns a floating point value of type `double`. If `str` cannot be converted to an expressible double value, `wcstod()` returns `HUGE_VAL`, defined in `math.h`, and sets `errno` to `ERANGE`

See Also "strtod" on page 337

wcstok

Description Extract tokens within a wide character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wchar_t * wcstok(wchar_t * str,
   const wchar_t * set);
```

Parameters Parameters for this function are:

str	wchar_t *	The string to be modified
set	wchar_t *	The list of character to find

Remarks Performs the same task as `strtok` for a wide character type.

Return When first called `wcstok()` returns a pointer to the first token in `str` or returns a null pointer if no token can be found.

Subsequent calls to `wcstok()` with a NULL `str` argument causes `wcstok()` to return a pointer to the next token or return a null pointer (`NULL`) when no more tokens exist.

See Also "strtok" on page 375

wcsftime

Description Formats a wide character string for time.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 392

wcsxfrm

Description Transform a locale-specific wide character array.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `dest` after it has received `source`.

See Also "strxfrm" on page 377

wctime

Description Convert a `time_t` type to a wide character array

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 386

wctrans

Description Constructs a property value for “toupper’ and “tolower” for character remapping.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
wctrans_t wctrans(const char *name);
```

Parameters Parameters for this function are:

name	const char *	toupper or tolower property

Remarks Constructs a value that represents a mapping between wide characters

Return A `wctrans_t` type

See Also “towctrans” on page 456

wmemchr

Description Search for an occurrence of a wide character.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `memchr` for a wide character type.

Return Returns a pointer to the found wide character, or a null pointer (`NULL`) if `val` cannot be found.

See Also "memchr" on page 346
"wcschr" on page 463

wmemcmp

Description Compare two blocks of memory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wchar.h>
int wmemcmp(const void * src1,
   const void * src2,
   size_t n);
```

Parameters Parameters for this function are:

src1	const void *	First string to compare
src2	const void *	Second string to compare
n	size_t	Maximum length to compare

Remarks Performs the same task as `memcmp` for a wide character type.

Return `wmemcmp` returns a zero if all `n` characters pointed to by `src1` and `src2` are equal.

`wmemcmp` returns a negative value if the first non-matching character pointed to by `src1` is less than the character pointed to by `src2`.

`wmemcmp` returns a positive value if the first non-matching character pointed to by `src1` is greater than the character pointed to by `src2`.

See Also "memcmp" on page 349

"wcscmp" on page 463

wmemcpy

Description Copy a contiguous memory block.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 350

wmemmove

Description Copy an overlapping contiguous memory block.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 351
"wcscpy" on page 465

wmemset

Description Clear the contents of a block of memory.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 352

wprintf

Description Send formatted wide character text to a standard output.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `printf` for a wide character type.

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 445

"printf" on page 269

"fwprintf" on page 447

wscanf

Description Reads a wide character formatted text from standard input.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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 `scanf` for a wide character type.

Return Returns the number of items successfully read and returns `WEOF` 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 445

"scanf" on page 284

"fwscanf" on page 450

34

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.

- "iswalnum," tests for alpha-numeric wide characters
- "iswalpha," tests for alphabetical wide characters
- "iswcntrl," tests for control wide characters
- "iswdigit," tests for digital wide characters
- "iswgraph," tests for graphical wide characters
- "iswlower," tests for lower wide characters
- "iswprint," tests for printable wide characters
- "iswpunct," tests for punctuation wide characters
- "iswspace," tests for whitespace wide characters
- "iswupper," tests for uppercase wide characters
- "iswxdigit," tests for a hexadecimal wide character type
- "towlower," converts wide characters to lower case
- "towupper," converts wide characters to upper case

iswalnum

Description Tests for alpha-numeric wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswalnum (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

Remarks Provides the same functionality as isalnum for wide character type.

Return True for an alphanumeric: `[a-z]`, `[A-Z]`, `[0-9]`

See Also "iswalnum"

iswalpha

Description Tests for alphabetical wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswalpha (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

Remarks Provides the same functionality as isalpha for wide character type.

Return True for an alphabetic: `[a-z]`, `[A-Z]`

See Also "iswalpha"

iswcntrl

Description Tests for control wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswcntrl (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

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

Description Tests for digital wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswdigit (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

Remarks Provides the same functionality as isdigit for wide character type.

Return True for a numeric character: `[0-9]`.

See Also "iswdigit"

iswgraph

Description Tests for graphical wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswgraph (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

Remarks Provides the same functionality as isgraph for wide character type.

Return True for a non-space printing character from the exclamation (`0x21`) to the tilde (`0x7E`).

See Also "iswgraph"

iswlower

Description Tests for lowercase wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswlower (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

Remarks Provides the same functionality as islower for wide character type.

Return True for a lowercase letter: `[a-z]`.

See Also "iswlower"

iswprint

Description Tests for printable wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswprint (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

Remarks Provides the same functionality as isprint for wide character type.

Return True for a printable character from space (`0x20`) to tilde (`0x7E`).

See Also "iswprint"

iswpunct

Description Tests for punctuation wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswpunct (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

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

Description Tests for whitespace wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswspace (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

Remarks Provides the same functionality as isspace for wide character type.

Return True for a space, tab, return, new line, vertical tab, or form feed.

See Also "isspace"

iswupper

Description Tests for uppercase wide characters.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswupper (wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

Remarks Provides the same functionality as isupper for wide character type.

Return True for an uppercase letter: `[A-Z]`.

See Also "isupper"

iswxdigit

Description Tests for a hexadecimal wide character type.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
int iswxdigit(wchar_t wc);
```

Parameters Parameters for this function are:

wc	wchar_t	The wide character to compare

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"

towlower

Description Converts wide characters from upper to lowercase.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

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

Description Converts wide characters from lower to uppercase.

Compatibility This function is compatible with the following targets:

ANSI	BeOS	EMB/RTOS	Mac OS	Palm OS	Win32	

Prototype

```
#include <wctype.h>
wchar_t towupper (wchar_t wc);
```

Parameters Parameters for this function are:

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"

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

T

U

V

W

CodeWarrior

MSL C Reference

Credits

writing lead: Ron Liechty

other writers: Marc Paquette

engineering: Vicki Scott and other MSL Engineers

frontline warriors: Tech Support and the entire MSL Team

Guide to CodeWarrior Documentation

CodeWarrior documentation is modular, like the underlying tools. There are manuals for the core tools, languages, libraries, and targets. The exact documentation provided with any CodeWarrior product is tailored to the tools included with the product. Your product will not have every manual listed here. However, you will probably have additional manuals (not listed here) for utilities or other software specific to your product.

Core Documentation	
IDE User Guide	How to use the CodeWarrior IDE
Debugger User Guide	How to use the CodeWarrior debugger
CodeWarrior Core Tutorials	Step-by-step introduction to IDE components
Language/Compiler Documentation	
C Compilers Reference	Information on the C/C++ front-end compiler
Pascal Compilers Reference	Information on the Pascal front-end compiler
Error Reference	Comprehensive list of compiler/linker error messages, with many fixes
Pascal Language Reference	The Metrowerks implementation of ANS Pascal
Assembler Guide	Stand-alone assembler syntax
Command-Line Tools Reference	Command-line options for Mac OS and Be compilers
Plugin API Manual	The CodeWarrior plugin compiler/linker API
Library Documentation	
MSL C Reference	Function reference for the Metrowerks ANSI standard C library
MSL C++ Reference	Function reference for the Metrowerks ANSI standard C++ library
Pascal Library Reference	Function reference for the Metrowerks ANS Pascal library
MFC Reference	Reference for the Microsoft Foundation Classes for Win32
Win32 SDK Reference	Microsoft's Reference for the Win32 API
The PowerPlant Book	Introductory guide to the Metrowerks application framework for Mac OS
PowerPlant Advanced Topics	Advanced topics in PowerPlant programming for Mac OS
Targeting Manuals	
Targeting BeOS	How to use CodeWarrior to program for BeOS
Targeting Java VM	How to use CodeWarrior to program for the Java Virtual Machine
Targeting Mac OS	How to use CodeWarrior to program for Mac OS
Targeting MIPS	How to use CodeWarrior to program for MIPS embedded processors
Targeting NEC V810/830	How to use CodeWarrior to program for NEC V810/830 processors
Targeting Net Yaroze	How to use CodeWarrior to program for Net Yaroze game console
Targeting Nucleus	How to use CodeWarrior to program for the Nucleus RTOS
Targeting OS-9	How to use CodeWarrior to program for the OS-9 RTOS
Targeting Palm OS	How to use CodeWarrior to program for PalmPilot
Targeting PlayStation OS	How to use CodeWarrior to program for the PlayStation game console
Targeting PowerPC Embedded Systems	How to use CodeWarrior to program for PPC embedded processors
Targeting VxWorks	How to use CodeWarrior to program for the VxWorks RTOS
Targeting Win32	How to use CodeWarrior to program for Windows