



Palm OS[®] Protein C/C++ Compiler Language & Library Reference

Palm OS[®] Developer Suite

CONTRIBUTORS

Written by Denise Stone.

Engineering contributions by Kenneth Albanowski, Matt Fassiotto, Ken Krugler, Kevin MacDonell, Vivek Magotra, Justin Morey, Jason Parks, Flash Sheridan, Phil Shoemaker, and Chris Tate.

Copyright © 2004, PalmSource, Inc. and its affiliates. All rights reserved. This technical documentation contains confidential and proprietary information of PalmSource, Inc. ("PalmSource"), and is provided to the licensee ("you") under the terms of a Nondisclosure Agreement, Product Development Kit license, Software Development Kit license or similar agreement between you and PalmSource. You must use commercially reasonable efforts to maintain the confidentiality of this technical documentation. You may print and copy this technical documentation solely for the permitted uses specified in your agreement with PalmSource. In addition, you may make up to two (2) copies of this technical documentation for archival and backup purposes. All copies of this technical documentation remain the property of PalmSource, and you agree to return or destroy them at PalmSource's written request. Except for the foregoing or as authorized in your agreement with PalmSource, you may not copy or distribute any part of this technical documentation in any form or by any means without express written consent from PalmSource, Inc., and you may not modify this technical documentation or make any derivative work of it (such as a translation, localization, transformation or adaptation) without express written consent from PalmSource.

PalmSource, Inc. reserves the right to revise this technical documentation from time to time, and is not obligated to notify you of any revisions.

THIS TECHNICAL DOCUMENTATION IS PROVIDED ON AN "AS IS" BASIS. NEITHER PALMSOURCE NOR ITS SUPPLIERS MAKES, AND EACH OF THEM EXPRESSLY EXCLUDES AND DISCLAIMS TO THE FULL EXTENT ALLOWED BY APPLICABLE LAW, ANY REPRESENTATIONS OR WARRANTIES REGARDING THIS TECHNICAL DOCUMENTATION, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION ANY WARRANTIES IMPLIED BY ANY COURSE OF DEALING OR COURSE OF PERFORMANCE AND ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, ACCURACY, AND SATISFACTORY QUALITY. PALMSOURCE AND ITS SUPPLIERS MAKE NO REPRESENTATIONS OR WARRANTIES THAT THIS TECHNICAL DOCUMENTATION IS FREE OF ERRORS OR IS SUITABLE FOR YOUR USE. TO THE FULL EXTENT ALLOWED BY APPLICABLE LAW, PALMSOURCE, INC. ALSO EXCLUDES FOR ITSELF AND ITS SUPPLIERS ANY LIABILITY, WHETHER BASED IN CONTRACT OR TORT (INCLUDING NEGLIGENCE), FOR DIRECT, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, EXEMPLARY OR PUNITIVE DAMAGES OF ANY KIND ARISING OUT OF OR IN ANY WAY RELATED TO THIS TECHNICAL DOCUMENTATION, INCLUDING WITHOUT LIMITATION DAMAGES FOR LOST REVENUE OR PROFITS, LOST BUSINESS, LOST GOODWILL, LOST INFORMATION OR DATA, BUSINESS INTERRUPTION, SERVICES STOPPAGE, IMPAIRMENT OF OTHER GOODS, COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR OTHER FINANCIAL LOSS, EVEN IF PALMSOURCE, INC. OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR IF SUCH DAMAGES COULD HAVE BEEN REASONABLY FORESEEN.

PalmSource, the PalmSource logo, BeOS, Graffiti, HandFAX, HandMAIL, HandPHONE, HandSTAMP, HandWEB, HotSync, the HotSync logo, iMessenger, MultiMail, MyPalm, Palm, the Palm logo, the Palm trade dress, Palm Computing, Palm OS, Palm Powered, PalmConnect, PalmGear, PalmGlove, PalmModem, Palm Pack, PalmPak, PalmPix, PalmPower, PalmPrint, Palm.Net, Palm Reader, Palm Talk, Simply Palm and ThinAir are trademarks of PalmSource, Inc. or its affiliates. All other product and brand names may be trademarks or registered trademarks of their respective owners.

IF THIS TECHNICAL DOCUMENTATION IS PROVIDED ON A COMPACT DISC, THE SOFTWARE AND OTHER DOCUMENTATION ON THE COMPACT DISC ARE SUBJECT TO THE LICENSE AGREEMENTS ACCOMPANYING THE SOFTWARE AND OTHER DOCUMENTATION.

Palm OS Protein C/C++ Compiler Language and Library Reference
Document Number 3124-001

June 29, 2004

For the latest version of this document, visit

<http://www.palmos.com/dev/support/docs/>

PalmSource, Inc.
1240 Crossman Avenue
Sunnyvale, CA 94089
USA

www.palmsource.com

Table of Contents

About This Book	vii
How This Book Is Organized	vii
Palm OS Developer Suite Documentation	viii
Additional Resources	ix

Part I: C/C++ Compiler Language Reference

1 Language Overview	3
C Technical Requirements	4
C++ Technical Requirements	4
Limitations	5
Restrictions on C99	5
Restrictions on C++.	5
2 Language Elements	7
Lexical Elements	7
Character Set	7
Comments	8
Tokens	9
Identifiers	9
Keywords	10
Constants	12
Operators	12
Separators.	15
Preprocessor Directives	15
#define	15
#pragma	17

Part II: C/C++ Compiler Library Reference

3 STLport/iostream	21
4 Palm OS-Specific Libraries	23
The Palm OS Implementation of the Standard C Library (libc)	24
5 Runtime Library Functions	25
Supported Functions	25
Unsupported Functions	31
6 assert.h	39
Functions and Macros	39
7 ctype.h	41
Functions and Macros	41
8 errno.h	49
Global Variables	49
9 fcntl.h	51
Functions and Macros	51
10 in.h	53
Structures and Types	53
Functions and Macros	54
11 inet.h	57
Functions and Macros	57
12 ioctl.h	63
Functions and Macros	63

13 iso646.h		65
14 locale.h		67
15 math.h		69
	Functions and Macros	71
16 netdb.h		109
	Structures and Types	109
	Functions and Macros	112
17 PalmMath.h		127
	Constants	127
	Functions and Macros	129
18 select.h		131
	Functions and Macros	131
19 socket.h		133
	Structures and Types	133
	Functions and Macros	134
20 stdarg.h		145
	Functions and Macros	145
21 stddef.h		147
	Functions and Macros	147
22 stdio.h		149
	Functions and Macros	149
23 stdlib.h		179
	Structures and Types	179
	Functions and Macros	181
24 string.h		197
	Functions and Macros	197

25 strings.h		213
	Functions and Macros 213
26 time.h		217
	Structures and Types 217
	Functions and Macros 218
27 time.h		229
	Constants 229
	Functions and Macros 229
28 uio.h		235
	Structures and Types 235
	Functions and Macros 235
29 unistd.h		237
	Functions and Macros 237
30 wchar.h		241
Index		243

About This Book

This book provides reference information about the C/C++ language and runtime libraries used with the Palm OS compiler tools. The audience for this book is application developers creating Palm OS Protein ARM-native applications and shared libraries using either the C or C++ programming languages for ARM-based handheld devices.

How This Book Is Organized

This book is divided into two parts, a language reference and a library reference.

[Part I, “C/C++ Compiler Language Reference,”](#) has the following organization:

- [Chapter 1, “Language Overview,”](#) on page 3 describes the technical requirements, language extensions, and limitations of the Palm OS compiler.
- [Chapter 2, “Language Elements,”](#) on page 7 describes the Palm OS compiler’s C/C++ language differences, as compared to the ANSI standard.

[Part II, “C/C++ Compiler Library Reference,”](#) has the following organization:

- [Chapter 3, “STLport/iostream,”](#) on page 21 describes the STLport implementation of the C++ standard template library.
- [Chapter 4, “Palm OS-Specific Libraries,”](#) on page 23 describes general library information.
- [Chapter 5, “Runtime Library Functions,”](#) on page 25 describes the supported and unsupported runtime functions.
- The chapters that follow, beginning with [Chapter 6, “assert.h,”](#) on page 39 each describe a specific header file and the supported structures, runtime functions, and macros defined within that header file.

About This Book

Palm OS Developer Suite Documentation

Palm OS Developer Suite Documentation

The following tools books are part of the Palm OS Developer Suite:

Document	Description
<i>Introduction to Palm OS Developer Suite</i>	Provides an overview of all of the Palm OS development tools: <ul style="list-style-type: none">• compiler tools• resource tools• testing and debugging tools
<i>Palm OS Protein C/C++ Compiler Tools Guide</i>	Describes how to use the Palm OS compiler tools: <ul style="list-style-type: none">• pacc – compiler• paasm – assembler• palink – linker• palib – librarian• PSLib – the Palm OS shared library tool, including information about shared library definition (SLD) files• PElf2Bin – Palm OS post linker• ElfDump – diagnostic tool
<i>Palm OS Resource Tools Guide</i>	Describes how to use the Palm OS resource tools: <ul style="list-style-type: none">• GenerateXRD – migration tool• Palm OS Resource Editor – XRD editor• PalmRC – building tool• PRCMerge – building tool• PRCCompare – comparison tool• hOverlay – localization tool• PRCSign and PRCCert – code-signing tools

Document	Description
<i>Palm OS Debugger Guide</i>	Describes how to use the Palm OS Debugger.
<i>Palm OS Resource File Formats</i>	Describes the XML formats used for XML resource definition (XRD) files. XRD files are used to define Palm OS resources and are the input files for the Palm OS resource tools.
<i>Palm OS Cobalt Simulator Guide</i>	Describes how to use the Palm OS Cobalt Simulator.
<i>Palm OS Virtual Phone Guide</i>	Describes how to use Virtual Phone.

Additional Resources

- Documentation
PalmSource publishes its latest versions of this and other documents for Palm OS developers at
<http://www.palmos.com/dev/support/docs/>
- Training
PalmSource and its partners host training classes for Palm OS developers. For topics and schedules, check
<http://www.palmos.com/dev/training>
- Knowledge Base
The Knowledge Base is a fast, web-based database of technical information. Search for frequently asked questions (FAQs), sample code, white papers, and development documentation at
<http://www.palmos.com/dev/support/kb/>

About This Book

Additional Resources



Part I

C/C++ Compiler

Language

Reference

This part is organized into the following chapters:

Language Overview	3
Language Elements	7

Language Overview

The Palm OS Protein C/C++ Compiler is a full-featured, standards-based, optimizing C/C++ compiler.

- The Palm OS compiler supports the C language standard ANSI/ISO/IEC 9899:1999, commonly known as C99, as a freestanding implementation. The compiler uses this language by default for C code.

It is required that you understand both the ANSI/ISO standard C language and library. The ANSI/ISO 9899:1999 C standards document completely describes the standard C library functions, as do several widely-used references including:

- *The C Programming Language*, Second Edition, by Brian W. Kernighan and Dennis M. Ritchie, Prentice Hall, Inc., 1988, ISBN 0-13-1103628.
 - *C: A Reference Manual*, Fifth Edition, by Samuel P. Harbison, Prentice Hall, Inc., 2002, ISBN 0-13-089592.
 - Online at www.dinkumware.com/refxc.html.
- The Palm OS compiler supports the C++ language standard ANSI/ISO/IEC 14882:1998(E). The C++ language standard is also documented in other widely-used references including *The C++ Programming Language*, Third Edition, by Bjarne Stroustrup, Addison-Wesley, 2000, ISBN 0-20-1700735.

The Palm OS compiler takes as input one or more C and/or C++ language text files (written according to the standards above) and produces a corresponding number of assembly language source files (see the [Palm OS Protein C/C++ Compiler Tools Guide](#) for more details).

C Technical Requirements

In addition to the ANSI/ISO/IEC requirements previously mentioned, the C facilities of the Palm OS compiler meet the following additional technical requirements:

- Supports a variety of useful extensions to the base language, particularly those useful to the ARM architecture.
- Supports compiling with extensions removed that are incompatible with the appropriate ANSI specification.
- Produces code for the ARM instruction set for version 4T architecture microprocessors as defined in the *ARM Reference Manual*, revision E.
- Adheres to the C calling conventions of the base standard ABI for the ARM architecture.
- Adheres to the shared library conventions documented in the *ARM-Thumb Shared Library Architecture* (ASHLA, document number MADEIRA-0020-CUST-DDES A-01).
- Produces DWARF version 1.1 debugging information, if debugging output is requested.

C++ Technical Requirements

In addition to the ANSI/ISO/IEC requirements previously mentioned, the C++ facilities of the Palm OS compiler meet the following additional technical requirements:

- Adheres to the C++ calling conventions of the base standard ABI for the ARM architecture.

Limitations

There are restrictions on some of the newer, more complex, and more exotic features of the relevant standards.

Restrictions on C99

The C99 implementation is limited in the following ways:

- Complex arithmetic is not supported, and thus all usages of the `_Complex` or `_Imaginary` types are unsupported. This includes:
 - `float _Complex`
 - `double _Complex`
 - `long double _Complex`
 - `float _Imaginary`
 - `double _Imaginary`
 - `long double _Imaginary`
- Avoid use of the `long double` type in the Simulator environment. It is unsupported and should not be used. There is a binary compatibility problem with i386 gcc and the compiler used to build the Simulator.
- Floating-point environment control is not available, therefore the `__STDC_IEC_559__` and `__STDC_IEC_559_COMPLEX__` macros are not defined.
- Variable length arrays are available, however during debugging, the array length may *not* be available. The allocation of local VLAs is implemented via calls to `malloc()` and `free()`.

Restrictions on C++

The C++ implementation is limited in the following way:

- Exported templates are not supported.

Language Overview

Limitations

Language Elements

This chapter describes the Palm OS compiler's C/C++ language differences, as compared to the ANSI standard. The following language elements of C and C++ are described:

- [Lexical Elements](#)
- [Preprocessor Directives](#)

Lexical Elements

This section describes the following lexical elements of C and C++:

- [Character Set](#)
- [Comments](#)
- [Tokens](#)
- [Identifiers](#)
- [Keywords](#)
- [Constants](#)
- [Operators](#)
- [Separators](#)

Character Set

The Palm OS compiler only specifically supports the ASCII character set for input, although the compiler is intended to be 8-bit neutral. The following lists the basic character set that is available at both compile and run time:

- The uppercase and lowercase letters of the English alphabet
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- The decimal digits 0 through 9
0 1 2 3 4 5 6 7 8 9

Language Elements

Lexical Elements

- The following graphic characters:

! " # % & ' () * + , - . / : ; < > ? [\] _
{ } ~

- The caret (^) character
- The split vertical bar (|) character
- The space character (' ')
- The control characters representing newline, horizontal tab, vertical tab, and form feed, and end of string (terminating null character).

The number sign (#) character is used for preprocessing only, and the underscore (_) character is treated as a normal letter.

Comments

The following comments within C/C++ source code are permitted:

- The /* (slash, asterisk) characters, followed by any sequence of characters (including newlines), followed by the */ (asterisk, slash) characters.
- The // (two slashes) characters followed by any sequence of characters. A newline not immediately preceded by a line-continuation (\) character terminates this form of comment. This kind of comment is commonly called a single-line comment.

You can put comments anywhere the language allows white space.

The Palm OS compiler also recognizes the following comments within C/C++ source code, used to affect warning messages generated by the compiler:

```
/*ARGSUSED*/
```

When placed before a function definition, this comment suppresses compiler warnings about unused parameters in functions.

```
/*NOTREACHED*/
```

When inserted at the beginning of a block of code that appears unreachable by the compiler, this comment suppresses the “unreachable code” warning.

Tokens

Source code is treated during preprocessing and compilation as a sequence of tokens. There are five different types of tokens:

- [Identifiers](#)
- [Keywords](#)
- [Constants](#)
- [Operators](#)
- [Separators](#)

Adjacent identifiers, keywords, and literals must be separated with white space. Other tokens should be separated by white space to make the source code more readable. White space includes blanks, horizontal and vertical tabs, newlines, form feeds, and comments.

Identifiers

An identifier consists of an arbitrary number of letters or digits; however, it must not begin with a digit and it must not have the same spelling as a keyword. Identifiers provide names for the following language elements:

- Functions
- Data objects
- Labels
- Enumerated tags
- Variables
- Macros
- Typedefs
- Structure members
- Union members

Keywords

Keywords are identifiers reserved by the language for special use.

- Refer to the C language standard: ANSI/ISO/IEC 9899:1999 specification for a list of the keywords common to the C language.
- Refer to the C++ language standard: ANSI/ISO/IEC 14882:1998 specification for a list of the keywords common to the C++ language.

Extension keywords

The Palm OS compiler also recognizes the following keywords:

`__align(n)`

n may be 1, 2, 4, 8, or 16. When applied to a global object, guarantees that the object is emitted with at least the specified alignment. When applied to a type declaration (e.g., `typedef` or `struct`), applies to all global objects that are instances of that type. *Note:* This keyword does not alter the packing within a structure or modify what code is used to access through a pointer. Use `__pack` or `#pragma pack` for the former, and `__packed` for the latter.

`asm`

The `asm` keyword is used to pass information through the compiler to the assembler. The Palm OS compiler permits assembler code to be inlined using the keywords `asm`, `_asm`, and `__asm`. The `asm` keyword has its normal C99 and C++ behavior; in addition, when used as the first keyword in a function definition, the contents of the function are all taken as assembly instructions and the function is emitted “naked,” without a prologue or epilogue that pushes or pops registers from the stack. (A ‘`bx lr`’ return instruction is placed after your code, in case you do not explicitly return.) An `asm` function is called in the same way as any function; its arguments are in registers `r0-r3` and on the stack, as is defined by ATPCS:

```
asm int func (int a, int b) {
add r0, r0, r1 // return a+b
}
```

The “inline” qualifier can be used with `asm` functions to indicate that the body of the `asm` function should be inserted

at each call-site. (The `asm` function should not explicitly return or use labels. As in the above example, it should fall off the end to return execution to the caller.)

Supported use of `asm` routines is limited to “nop,” as an inline `asm` statement and relatively small `asm` functions that do not use labels.

`__asm`

Followed by curly brackets, indicates a multi-line inline assembly block. Otherwise, indicates inline assembly until the end of the current line.

`__inline`

An exact alias of the normal `inline` keyword, in C99 or C++, depending on which is being compiled.

`__int64`

Alias for `long long` type.

`__pack(n)`

n may be 0, 1, 2, 4, 8, or 16. Applied to a structure definition, this keyword changes the packing in effect for that structure. This keyword overrides any `#pragma pack()` setting for this structure. If zero (0) is selected, natural alignment is used (not the current `#pragma pack` value).

`__packed`

Hybrid modifier: when applied to a structure definition, forces the packing to be 1-byte aligned. When applied to a pointer, forces all accesses through that pointer to assume an unaligned pointer. (This is also the case when a pointer to a `__packed` structure is used.)

`__pure`

In function prototypes modifying the function name, this keyword indicates that the function has no side-effects and relies only on its input parameters. Currently, the Palm OS compiler ignores this keyword.

`__ror32(x, y)`

A built-in operator that returns the 32-bit unsigned integer *x* rotated right by *y* bits.

`__value_in_regs`

When this keyword is applied to a function prototype or declaration, states that the return value of the function, if it is

Language Elements

Lexical Elements

a small structure (16 bytes or less), is passed in processor registers r0-r3. (Normally structure return values are passed by pointer in a hidden first argument.)

This calling convention keyword is potentially useful to interoperate with special routines.

Example:

```
struct div_result {int div, rem;};  
struct div_result __value_in_regs do_div  
(int x, int y);
```

`__weak`

In declarations of external objects (functions or data), this modifier indicates that the object is not required and the linker should fix up references if the object is not available during linkage.

Constants

The value of any constant must be in the range of representable values for its type. The C language contains the following types of constants (also called *literals*):

- Integer (decimal, octal, or hexadecimal notation)
- Floating-point (`double`, `float`, `long double`, or hexadecimal notation)
- Character (one or more characters in apostrophes)
- String (sequence of characters enclosed in double quotes)
- Enumeration

Operators

Operators can be classified as:

- [Postfix](#)
- [Prefix](#)
- [Normal](#)
- [Boolean](#)
- [Assignment](#)
- [C++ Compatibility](#)

Postfix

Postfix operators are operators that are suffixed to an expression, such as, `operand++`.

Prefix

Prefix operators are operators that are prefixed to an expression, such as, `++operand` or `!operand`.

Normal

There are several normal operators that return the result defined for each:

<code>+</code>	addition
<code>-</code>	subtraction
<code>*</code>	multiplication
<code>/</code>	division
<code>%</code>	modulo
<code>&</code>	AND
<code> </code>	OR
<code>^</code>	XOR
<code>>></code>	shift right
<code><<</code>	shift left

Boolean

The Boolean operators return either 1 (true) or 0 (false).

<code>&&</code>	logical AND
<code> </code>	logical OR
<code><</code>	less than
<code>></code>	greater than
<code><=</code>	less than equal
<code>>=</code>	greater than equal
<code>==</code>	equal
<code>!=</code>	not equal

Language Elements

Lexical Elements

Assignment

An assignment operator stores the value of the right expression into the left expression:

= a = b assigns the value of b into a
*= a *= b is equivalent to a = a * b
/= a /= b is equivalent to a = a / b
%= a %= b is equivalent to a = a % b
+= index += 2 is equivalent to index = index + 2
-= index -= 3 is equivalent to index = index - 3
<<= n1 <<= n2 is equivalent to n1 = n1 << n2
>>= n1 >>= n2 is equivalent to n1 = n1 >> n2
&= mask &= 2 is equivalent to mask = mask & 2
^= t1 ^= t2 is equivalent to t1 = t1 ^ t2
|= flag |= ON is equivalent to flag = flag | ON

C++ Compatibility

There are three new compound operators in C++:

- .* Binds its second operand, which shall be of type "pointer to member of T" (where T is a completely defined class type) to its first operand, which shall be of class T.
- >* Binds its second operand, which shall be of type "pointer to member of T" (where T is a completely defined class type) to its first operand, which shall be of type "pointer to T" or "pointer to a class of which T is an unambiguous and accessible base class."
- :: Allows a type, an object, a function, an enumerator, or a namespace declared in the global namespace to be referred to even if its identifier has been hidden.

Separators

Separators can include:

- () parenthesis
- [] brackets
- { } braces
- ,
- ;
- :

Preprocessor Directives

Preprocessor directives instruct the preprocessor to act on the text of the program. Preprocessor directives begin with the # token followed by a preprocessor keyword. The # token must appear as the first character that is not white space on a line. The # is not part of the directive name and can be separated from the name with white space. Except for some #pragma directives, preprocessor directives can appear anywhere in a program.

#define

A *preprocessor define directive* directs the preprocessor to replace all subsequent occurrences of a macro with specified replacement tokens. This section describes the #define commands that the Palm OS compiler recognizes.

__APGE__
Defined as 1.

__APOGEE__
Defined as 1.

__arm
Defined as 1.

__BOOL
Defined in C++ mode when bool is a keyword.

__cplusplus
Defined in C++ mode.

Language Elements

Preprocessor Directives

- `c_plusplus`
Defined in default C++ mode, but not in `strict` mode.
- `__DATE__`
Defined in all modes to the date of the compilation in the form "Mmm dd yyyy."
- `__EDG__`
Always defined.
- `__EDG_VERSION__`
Defined to an integral value that represents the version number of the front end. For example, version 2.30 is represented as 230.
- `__embedded_cplusplus`
Defined as 1 in embedded C++ mode.
- `__EXCEPTIONS`
Defined in C++ mode when exception handling is enabled.
- `__PACC_VER`
0xMmmrrbbb, where (M=Major, m=minor, r=rev, b=build). For example, 0x1000000D, for 1.0.0.13.
- `__PALMSOURCE__`
Defined as 1.
- `__PSI__`
Defined as 1.
- `__RTTI`
Defined in C++ mode when RTTI is enabled.
- `__SIGNED_CHARS__`
Defined when plain character is signed. (By default, the character type is unsigned.)
- `__STDC__`
Defined in ANSI C mode and in C++ mode. In C++ mode, the value may be redefined.
- `__STDC_HOSTED__`
Defined in C99 mode with the value zero (0).
- `__STDC_VERSION__`
Defined in ANSI C mode with the value 199901L.

`__TIME__`

Defined in all modes to the time of the compilation in the form "hh:mm:ss."

`_WCHAR_T`

Defined in C++ mode when `wchar_t` is a keyword.

#pragma

A *pragma directive* is an implementation-defined instruction to the compiler. This section describes the `#pragma` commands that the Palm OS compiler recognizes.

`#pragma once`

Indicates that a source file (usually a header) need not be included again. (Thus an `#include` of the same header has no effect.) If normal header guards are used, the compiler optimizes them into a `#pragma once`:

```
#pragma once // unnecessary
#ifndef MY_HEADER_GUARD
#define MY_HEADER_GUARD
// header contents ...
#endif /* MY_HEADER_GUARD */
```

`#pragma pack(n)`

Sets current structure packing to *n*, where *n* is 1, 2, 4, 8, or 16.

`#pragma pack()`

Resets current structure packing to natural alignment.

`#pragma pack (pop [,name] [,n])`

If *name* is supplied, pops back to the position on the stack with that *name*, otherwise pops a single value off the stack. If *n* is supplied, sets the alignment to that value after popping.

`#pragma pack (push [,name] [,n])`

Pushes the current structure packing onto a stack. If *name* (an identifier) is supplied, names the prior position on the stack. If *n* is supplied, sets the packing to that value, after pushing the original value.

`#pragma weak name`

Same as declaring the global object with the external name of *name* with the `__WEAK` attribute.

Language Elements

Preprocessor Directives



Part II

C/C++ Compiler Library Reference

This part is organized in the following manner: general library and runtime function information appears first, followed by detailed header file information that documents the supported structures, runtime functions, and macros. Note that header file chapters, which are organized alphabetically, follow the “[Runtime Library Functions](#)” chapter, which overviews the supported runtime functions provided by the operating system and the *unsupported* runtime functions not implemented by Palm OS.

STLport/iostream	21
Palm OS-Specific Libraries	23
Runtime Library Functions	25
assert.h	39
ctype.h	41
errno.h	49
fcntl.h	51
in.h	53
inet.h	57
ioctl.h	63
iso646.h	65
locale.h	67
math.h	69
netdb.h	109
PalmMath.h	127

<u>select.h</u>	131
<u>socket.h</u>	133
<u>stdarg.h</u>	145
<u>stddef.h</u>	147
<u>stdio.h</u>	149
<u>stdlib.h</u>	179
<u>string.h</u>	197
<u>strings.h</u>	213
<u>time.h</u>	217
<u>time.h</u>	229
<u>uio.h</u>	235
<u>unistd.h</u>	237
<u>wchar.h</u>	241

STLport/iostream

The Palm OS Protein C/C++ Compiler Suite includes and supports the STLport implementation of the C++ standard template library.

Specific details regarding the implementation of the C++ STLport/iostream material is not currently documented in this manual; for documentation, visit <http://www.stlport.org/doc/index.html>. However, the following information may be useful:

- iostreams are implemented in terms of `stdio`; `cout` is connected to `stdout`, `cerr` is connected to `stderr`, and `cin` is connected to `stdin`.
- no locale functionality beyond the C locale is supported.
- all other pieces of STL functionality are believed to be supported.

For more information on the functionality provided by the C++ standard library, please consult documentation on the C++ language, such as *The C++ Programming Language*, Third Edition, by Bjarne Stroustrup, or the ANSI/ISO specification, available as ANSI/ISO/IEC document 14882:1998.

Palm OS-Specific Libraries

An integral part of the Palm OS Protein C/C++ Compiler are the standard headers, startup code, and run-time libraries. The supplied run-time libraries serve several purposes:

- `cpp` — The `cpp` libraries implement objects common to any C++ standard library (e.g., the standard exception objects).
- `eabi` — The `eabi` libraries implement preliminary ARM EABI support on top of Palm OS. They implement the necessary EABI support routines, translating them into Palm OS specific routine calls.
- `pacc` — The `pacc` libraries implement objects and routines that are unique or particular to the Palm OS compiler and are not required or useful with any other tool chain.
- `STLport` — The C++ standard template library features thread safety, improved memory utilization, improved run-time efficiency, and new data structures, including hash tables.
- `support` — This is an implementation of the floating-point and integral support functions. The Palm OS compiler automatically links with this library, however, the `FloatMgr` library should also be linked.

The Palm OS Implementation of the Standard C Library (libc)

The Palm OS implementation of the standard C library is derived from the NetBSD ARM source base, with some modification due to the non-Unix nature of Palm OS:

- In the future, it may be possible to direct `stdout/stdin` operations through other I/O devices; no timeline for this has been stated.
- The C99 header `<complex.h>` is not supported in this version of `libc`. Applications using complex numbers should use STLport or another ANSI compliant C++ library.
- The C99 header `<fenv.h>` is not supported in this version of `libc`. MathLib does not raise floating exceptions and does not respond to varying rounding modes. Checking `errno`, and checking the return value can handle exceptional cases.
- There is also no `<setjmp.h>` implementation. The `<ErrTryCatch.h>` header can provide much of the same functionality, but the standard C interface is not yet supported.
- In addition, the following POSIX header files are not documented in this reference because they are either fairly self-explanatory or do not contain any runtime library functions that are provided by the operating system.

- `<climits.h>`
- `<inttypes.h>`
- `<limits.h>`
- `<namespace.h>`
- `<paths.h>`
- `<signal.h>`
- `<stdint.h>`
- `<termios.h>`

Runtime Library Functions

Supported Functions

The following is an alphabetical list of runtime library functions, as defined in the POSIX headers for Palm OS 6.0.1, which are explicitly provided by the operating system. For detailed information about any of these functions, see the individual header file chapters that follow, beginning with [Chapter 6](#), “[assert.h](#).”

Table 5.1 **posix/ctype.h**

isalnum()	isgraph()	isupper()
isalpha()	islower()	isxdigit()
isblank()	isprint()	tolower()
iscntrl()	ispunct()	toupper()
isdigit()	isspace()	

Table 5.2 **posix/fcntl.h**

fcntl()	open()
-------------------------	------------------------

Table 5.3 **posix/math.h**

abs()	expf()	logf()
acos()	expl()	logl()

Runtime Library Functions

Supported Functions

Table 5.3 `posix/math.h` (continued)

<u>acosf()</u>	<u>expm1()</u>	<u>modf()</u>
<u>acosh()</u>	<u>fabs()</u>	<u>modff()</u>
<u>acosl()</u>	<u>fabsf()</u>	<u>modfl()</u>
<u>asin()</u>	<u>fabsl()</u>	<u>nextafter()</u>
<u>asinf()</u>	<u>floor()</u>	<u>pow()</u>
<u>asinh()</u>	<u>floorf()</u>	<u>powf()</u>
<u>asinl()</u>	<u>floorl()</u>	<u>powl()</u>
<u>atan()</u>	<u>fmod()</u>	<u>remainder()</u>
<u>atan2()</u>	<u>fmodf()</u>	<u>rint()</u>
<u>atan2f()</u>	<u>fmodl()</u>	<u>scalbn()</u>
<u>atan2l()</u>	<u>frexp()</u>	<u>sin()</u>
<u>atanf()</u>	<u>frexpf()</u>	<u>sinf()</u>
<u>atanh()</u>	<u>frexpl()</u>	<u>sinh()</u>
<u>atanl()</u>	<u>hypot()</u>	<u>sinhf()</u>
<u>cbrt()</u>	<u>hypotf()</u>	<u>sinhl()</u>
<u>ceil()</u>	<u>hypotl()</u>	<u>sinl()</u>
<u>ceilf()</u>	<u>ilogb()</u>	<u>sqrt()</u>
<u>ceill()</u>	<u>ldexp()</u>	<u>sqrtf()</u>
<u>copysign()</u>	<u>ldexpf()</u>	<u>sqrtl()</u>
<u>cos()</u>	<u>ldexpl()</u>	<u>tan()</u>
<u>cosf()</u>	<u>log()</u>	<u>tanf()</u>
<u>cosh()</u>	<u>log10()</u>	<u>tanh()</u>
<u>coshf()</u>	<u>log10f()</u>	<u>tanhf()</u>
<u>coshl()</u>	<u>log10l()</u>	<u>tanhl()</u>

Table 5.3 **posix/math.h** (*continued*)

<u>cosl()</u>	<u>loglp()</u>	<u>tanl()</u>
<u>exp()</u>	<u>logb()</u>	

Table 5.4 **posix/netdb.h**

<u>endhostent()</u>	<u>gethostbyname2()</u>	<u>getprotoent()</u>
<u>endnetent()</u>	<u>gethostent()</u>	<u>getservbyname()</u>
<u>endprotoent()</u>	<u>getipnodebyaddr()</u>	<u>getservbyport()</u>
<u>endservent()</u>	<u>getipnodebyname()</u>	<u>getservent()</u>
<u>freeaddrinfo()</u>	<u>getnameinfo()</u>	<u>hstrerror()</u>
<u>freehostent()</u>	<u>getnetbyaddr()</u>	<u>sethostent()</u>
<u>gai_strerror()</u>	<u>getnetbyname()</u>	<u>setnetent()</u>
<u>getaddrinfo()</u>	<u>getnetent()</u>	<u>setprotoent()</u>
<u>gethostbyaddr()</u>	<u>getprotobyname()</u>	<u>setservent()</u>
<u>gethostbyname()</u>	<u>getprotobynumber()</u>	

Table 5.5 **posix/stdio.h**

<u>asprintf()</u>	<u>freopen()</u>	<u>rewind()</u>
<u>clearerr()</u>	<u>fscanf()</u>	<u>scanf()</u>
<u>fclose()</u>	<u>fseek()</u>	<u>setbuf()</u>
<u>fdopen()</u>	<u>fseeko()</u>	<u>setbuffer()</u>
<u>feof()</u>	<u>fsetpos()</u>	<u>setlinebuf()</u>
<u>ferror()</u>	<u>ftell()</u>	<u>setvbuf()</u>
<u>fflush()</u>	<u>ftello()</u>	<u>snprintf()</u>
<u>fgetc()</u>	<u>fwrite()</u>	<u>sprintf()</u>

Runtime Library Functions

Supported Functions

Table 5.5 `posix/stdio.h` (continued)

<code>fgetln()</code>	<code>getc()</code>	<code>sscanf()</code>
<code>fgetpos()</code>	<code>getchar()</code>	<code>ungetc()</code>
<code>fgets()</code>	<code>gets()</code>	<code>vasprintf()</code>
<code>fileno()</code>	<code>getw()</code>	<code>vfprintf()</code>
<code>fopen()</code>	<code>perror()</code>	<code>vprintf()</code>
<code>fprintf()</code>	<code>printf()</code>	<code>vscanf()</code>
<code>fpurge()</code>	<code>putc()</code>	<code>vsnprintf()</code>
<code>fputc()</code>	<code>putchar()</code>	<code>vsprintf()</code>
<code>fputs()</code>	<code>puts()</code>	<code>vsscanf()</code>
<code>fread()</code>	<code>putw()</code>	

Table 5.6 `posix/stdlib.h`

<code>abs()</code>	<code>labs()</code>	<code>realloc()</code>
<code>atof()</code>	<code>ldiv()</code>	<code>setenv()</code>
<code>atoi()</code>	<code>llabs()</code>	<code>srand()</code>
<code>atol()</code>	<code>malloc()</code>	<code>srandom()</code>
<code>atoll()</code>	<code>putenv()</code>	<code>strtod()</code>
<code>bsearch()</code>	<code>qsort()</code>	<code>strtol()</code>
<code>calloc()</code>	<code>qsort_r()</code>	<code>strtoll()</code>
<code>div()</code>	<code>rand()</code>	<code>strtoul()</code>
<code>free()</code>	<code>rand_r()</code>	<code>strtoull()</code>
<code>getenv()</code>	<code>random()</code>	<code>unsetenv()</code>
<code>inplace_realloc()</code>		

Table 5.7 **posix/string.h**

<u>memchr()</u>	<u>strcspn()</u>	<u>strncpy()</u>
<u>memcmp()</u>	<u>strdup()</u>	<u>strpbrk()</u>
<u>memcpy()</u>	<u>strerror()</u>	<u>strrchr()</u>
<u>memmove()</u>	<u>strerror_r()</u>	<u>strsep()</u>
<u>memset()</u>	<u>strlcat()</u>	<u>strspn()</u>
<u>strcat()</u>	<u>strncpy()</u>	<u>strstr()</u>
<u>strchr()</u>	<u>strlen()</u>	<u>strtok()</u>
<u>strcmp()</u>	<u>strncat()</u>	<u>strtok_r()</u>
<u>strcoll()</u>	<u>strncmp()</u>	<u>strxfrm()</u>
<u>strcpy()</u>		

Table 5.8 **posix/strings.h**

<u>bcopy()</u>	<u>strcasecmp()</u>
<u>bzero()</u>	<u>strncasecmp()</u>

Table 5.9 **posix/time.h**

<u>asctime()</u>	<u>difftime()</u>	<u>mktime()</u>
<u>asctime_r()</u>	<u>gmtime()</u>	<u>strftime()</u>
<u>clock()</u>	<u>gmtime_r()</u>	<u>time()</u>
<u>ctime()</u>	<u>localtime()</u>	<u>time()</u>
<u>ctime_r()</u>	<u>localtime_r()</u>	

Runtime Library Functions

Supported Functions

Table 5.10 posix/unistd.h

<u>close()</u>	<u>isatty()</u>	<u>write()</u>
<u>getopt()</u>	<u>read()</u>	

Table 5.11 posix/arpa/inet.h

<u>inet_addr()</u>	<u>inet_makeaddr()</u>	<u>inet_ntoa()</u>
<u>inet_aton()</u>	<u>inet_netof()</u>	<u>inet_ntop()</u>
<u>inet_lnaof()</u>	<u>inet_network()</u>	<u>inet_pton()</u>

Table 5.12 posix/netinet/in.h

<u>htonl()</u>	<u>ntohl()</u>
<u>htons()</u>	<u>ntohl()</u>

Table 5.13 posix/sys/ioctl.h

<u>ioctl()</u>

Table 5.14 posix/sys/PalmMath.h

<u>lceilf()</u>	<u>lfloorf()</u>	<u>sincosf()</u>
---------------------------------	----------------------------------	----------------------------------

Table 5.15 posix/sys/select.h

<u>select()</u>

Table 5.16 `posix/sys/socket.h`

<code>accept()</code>	<code>listen()</code>	<code>sendmsg()</code>
<code>bind()</code>	<code>recv()</code>	<code>sendto()</code>
<code>connect()</code>	<code>recvfrom()</code>	<code>setsockopt()</code>
<code>getpeername()</code>	<code>recvmsg()</code>	<code>shutdown()</code>
<code>getsockname()</code>	<code>send()</code>	<code>socket()</code>
<code>getsockopt()</code>		

Table 5.17 `posix/sys/time.h`

<code>getcountrycode()</code>	<code>palm_seconds_to_time_t()</code>
<code>getgmtoffset()</code>	<code>settime()</code>
<code>gettimezone()</code>	<code>settimezone()</code>
<code>hastimezone()</code>	<code>system_real_time()</code>
<code>localtime_tz()</code>	<code>system_time()</code>
<code>mktime_tz()</code>	<code>time_t_to_palm_seconds()</code>

Table 5.18 `posix/sys/uio.h`

<code>readv()</code>	<code>writev()</code>
--------------------------------------	---------------------------------------

Unsupported Functions

The following is an alphabetical list of runtime library functions, sorted by header file name, declared in the POSIX headers that are *not* implemented by the operating system.

Runtime Library Functions

Unsupported Functions

Table 5.19 `posix/ctype.h`

<code>isascii()</code> (this is handled via a <code>#define</code>)	<code>toascii()</code> (this is handled via a <code>#define</code>)
---	---

Table 5.20 `posix/inttypes.h`

<code>strtoimax()</code>	<code>strtoumax()</code>
--------------------------	--------------------------

Table 5.21 `posix/locale.h`

<code>setlocale()</code>

Table 5.22 `posix/math.h`

<code>erf()</code>	<code>islessequal()</code>	<code>modf()</code>
<code>erfc()</code>	<code>islessgreater()</code>	<code>nan()</code>
<code>exp2()</code>	<code>isunordered()</code>	<code>nearbyint()</code>
<code>fdim()</code>	<code>lgamma()</code>	<code>nexttoward()</code>
<code>fma()</code>	<code>llrint()</code>	<code>remquo()</code>
<code>fmax()</code>	<code>llround()</code>	<code>round()</code>
<code>fmin()</code>	<code>log2()</code>	<code>scalbln()</code>
<code>isgreater()</code>	<code>lrint()</code>	<code>tgamma()</code>
<code>isgreaterequal()</code>	<code>lround()</code>	<code>trunc()</code>
<code>isless()</code>		

In addition, any of the above functions that have `float` overrides (suffixed with an “f”) or `long double` overrides (suffixed with an “l”) are also unsupported. For example, `exp2f()` and `exp2l()`.

Table 5.23 `posix/signal.h`

<code>kill()</code>	<code>sigblock()</code>	<code>sigpending()</code>
<code>killpg()</code>	<code>sigdelset()</code>	<code>sigprocmask()</code>
<code>psignal()</code>	<code>sigemptyset()</code>	<code>sigreturn()</code>
<code>raise()</code>	<code>sigfillset()</code>	<code>sigsetmask()</code>
<code>sigaction()</code>	<code>siginterrupt()</code>	<code>sigstack()</code>
<code>sigaddset()</code>	<code>sigismember()</code>	<code>sigsuspend()</code>
<code>sigaltstack()</code>	<code>sigpause()</code>	<code>sigvec()</code>

Table 5.24 `posix/stdio.h`

<code>ctermid()</code>	<code>getc_unlocked()</code>	<code>remove()</code>
<code>cuserid()</code>	<code>getchar_unlocked()</code>	<code>rename()</code>
<code>flockfile()</code>	<code>pclose()</code>	<code>tempnam()</code>
<code>ftrylockfile()</code>	<code>popen()</code>	<code>tmpfile()</code>
<code>funlockfile()</code>	<code>putc_unlocked()</code>	<code>tmpnam()</code>
<code>funopen()</code>	<code>putchar_unlocked()</code>	

Table 5.25 `posix/stdlib.h`

<code>a64l()</code>	<code>daemon()</code>	<code>mkdtemp()</code>
<code>abort()</code>	<code>devname()</code>	<code>mkstemp()</code>
<code>alloca()</code>	<code>drand48()</code>	<code>mktemp()</code>
<code>atexit()</code>	<code>erand48()</code>	<code>rand48()</code>
<code>cfree()</code>	<code>exit()</code>	<code>rand48()</code>
<code>cgetcap()</code>	<code>getbsize()</code>	<code>qdiv()</code>

Runtime Library Functions

Unsupported Functions

Table 5.25 `posix/stdlib.h` (*continued*)

<code>cgetc</code>	<code>getloadavg</code>	<code>radixsort</code>
<code>cgetent</code>	<code>heapsort</code>	<code>realpath</code>
<code>cgetfirst</code>	<code>initstate</code>	<code>seed48</code>
<code>cgetmatch</code>	<code>jrand48</code>	<code>setkey</code>
<code>cgetnext</code>	<code>l64a</code>	<code>setstate</code>
<code>cgetnum</code>	<code>lcong48</code>	<code>sradixsort</code>
<code>cgetset</code>	<code>lldiv</code>	<code>srand48</code>
<code>cgetstr</code>	<code>lrand48</code>	<code>ttyslot</code>
<code>cgetustr</code>	<code>mergesort</code>	<code>valloc</code>

Table 5.26 `posix/string.h`

<code>memccpy</code>

Table 5.27 `posix/strings.h`

<code>bcmp</code>	<code>index</code>
<code>ffs</code>	<code>rindex</code>

Table 5.28 `posix/termios.h`

<code>tcdrain</code>	<code>tcflush</code>	<code>tcsendbreak</code>
<code>tcflow</code>	<code>tcgetpgrp</code>	<code>tcsetpgrp</code>

Table 5.29 `posix/time.h`

<code>clock_getres()</code>	<code>strptime()</code>	<code>timer_getoverrun()</code>
<code>clock_gettime()</code>	<code>time2posix()</code>	<code>timer_gettime()</code>
<code>clock_settime()</code>	<code>timelocal()</code>	<code>timer_settime()</code>
<code>nanosleep()</code>	<code>timeoff()</code>	<code>timezone()</code>
<code>offtime()</code>	<code>timer_create()</code>	<code>tzset()</code>
<code>posix2time()</code>	<code>timer_delete()</code>	<code>tzsetwall()</code>

Table 5.30 `posix/unistd.h`

<code>access()</code>	<code>getpass()</code>	<code>setdomainname()</code>
<code>acct()</code>	<code>getpgid()</code>	<code>setegid()</code>
<code>alarm()</code>	<code>getpgrp()</code>	<code>seteuid()</code>
<code>brk()</code>	<code>getpid()</code>	<code>setgid()</code>
<code>chdir()</code>	<code>getppid()</code>	<code>setgroups()</code>
<code>chown()</code>	<code>getsid()</code>	<code>sethostid()</code>
<code>chroot()</code>	<code>getsubopt()</code>	<code>sethostname()</code>
<code>confstr()</code>	<code>getuid()</code>	<code>setlogin()</code>
<code>crypt()</code>	<code>getusershell()</code>	<code>setmode()</code>
<code>cuserid()</code>	<code>getwd()</code>	<code>setpgid()</code>
<code>des_cipher()</code>	<code>initgroups()</code>	<code>setpgrp()</code>
<code>des_setkey()</code>	<code>iruserok()</code>	<code>setregid()</code>
<code>dup()</code>	<code>iruserok_sa()</code>	<code>setreuid()</code>
<code>dup2()</code>	<code>issetugid()</code>	<code>setrgid()</code>
<code>encrypt()</code>	<code>lchown()</code>	<code>setruid()</code>

Runtime Library Functions

Unsupported Functions

Table 5.30 `posix/unistd.h` (*continued*)

<code>endusershell()</code>	<code>link()</code>	<code>setsid()</code>
<code>exec()</code>	<code>lockf()</code>	<code>setuid()</code>
<code>fchdir()</code>	<code>lseek()</code>	<code>setusershell()</code>
<code>fchown()</code>	<code>nfssvc()</code>	<code>sleep()</code>
<code>fchroot()</code>	<code>nice()</code>	<code>strmode()</code>
<code>fdatasync()</code>	<code>pathconf()</code>	<code>strsignal()</code>
<code>fpathconf()</code>	<code>pause()</code>	<code>swab()</code>
<code>fsync()</code>	<code>pread()</code>	<code>swapctl()</code>
<code>ftruncate()</code>	<code>profil()</code>	<code>swapon()</code>
<code>getcwd()</code>	<code>psignal()</code>	<code>symlink()</code>
<code>getdomainname()</code>	<code>pwrite()</code>	<code>sync()</code>
<code>getdtablesize()</code>	<code>rcmd()</code>	<code>syscall()</code>
<code>getegid()</code>	<code>rcmd_af()</code>	<code>sysconf()</code>
<code>geteuid()</code>	<code>readlink()</code>	<code>tcgetpgrp()</code>
<code>getgid()</code>	<code>reboot()</code>	<code>tcsetpgrp()</code>
<code>getgrouplist()</code>	<code>rename()</code>	<code>truncate()</code>
<code>getgroups()</code>	<code>revoke()</code>	<code>ttyname()</code>
<code>gethostid()</code>	<code>rmdir()</code>	<code>ualarm()</code>
<code>gethostname()</code>	<code>rresvport()</code>	<code>undelete()</code>
<code>getlogin()</code>	<code>rresvport_af()</code>	<code>unlink()</code>
<code>getmode()</code>	<code>ruserok()</code>	<code>usleep()</code>
<code>getpagesize()</code>	<code>sbrk()</code>	<code>vfork()</code>

Table 5.31 `posix/wchar.h`

<code>fwide()</code>	<code>wcsncat()</code>	<code>wcstoul()</code>
<code>wscat()</code>	<code>wcsncmp()</code>	<code>wcswidth()</code>
<code>wchr()</code>	<code>wcsncpy()</code>	<code>wcwidth()</code>
<code>wscmp()</code>	<code>wcspbrk()</code>	<code>wmemchr()</code>
<code>wscopy()</code>	<code>wcsrchr()</code>	<code>wmemcmp()</code>
<code>wscspn()</code>	<code>wcsspn()</code>	<code>wmemcpy()</code>
<code>wslcat()</code>	<code>wcsstr()</code>	<code>wmemmove()</code>
<code>wslcpy()</code>	<code>wcstod()</code>	<code>wmemset()</code>
<code>wslen()</code>	<code>wcstol()</code>	

Table 5.32 `posix/machine/arm/param.h`

<code>delay()</code>

Table 5.33 `posix/sys/bswap.h`

<code>bswap16()</code>	<code>bswap32()</code>	<code>bswap64()</code>
------------------------	------------------------	------------------------

Table 5.34 `posix/sys/socket.h`

<code>socketpair()</code>

Table 5.35 `posix/sys/stat.h`

<code>chflags()</code>	<code>lchflags()</code>	<code>mkfifo()</code>
<code>chmod()</code>	<code>lchmod()</code>	<code>mknod()</code>

Runtime Library Functions

Unsupported Functions

Table 5.35 `posix/sys/stat.h` (*continued*)

<code>fchflags()</code>	<code>lstat()</code>	<code>stat()</code>
<code>fchmod()</code>	<code>mkdir()</code>	<code>umask()</code>
<code>fstat()</code>		

Table 5.36 `posix/sys/time.h`

<code>adjtime()</code>	<code>itimerdecr()</code>	<code>ratecheck()</code>
<code>adjtime1()</code>	<code>itimerfix()</code>	<code>setitimer()</code>
<code>clock_settime1()</code>	<code>lutimes()</code>	<code>settimeofday()</code>
<code>futimes()</code>	<code>microtime()</code>	<code>settimeofday1()</code>
<code>getitimer()</code>	<code>ppsratecheck()</code>	<code>utimes()</code>
<code>gettimeofday()</code>		

Table 5.37 `posix/sys/uio.h`

<code>preadv()</code>	<code>pwritev()</code>
-----------------------	------------------------

assert.h

The `<assert.h>` header defines the `assert ()` macro, which is used for debugging purposes. It also refers to another macro, `NDEBUG`, which is defined elsewhere.

Functions and Macros

assert Macro

Purpose	Outputs a diagnostic message to the standard error file and stops the program if a test fails.
Prototype	<code>assert (condition)</code>
Parameters	<code>→ condition</code> Diagnostic information.
Comments	If <code>condition</code> is true, the <code>assert ()</code> macro does nothing.

assert.h

assert

ctype.h

The `<ctype.h>` header defines several functions useful for classifying and converting characters.

Functions and Macros

isalnum Function

Purpose	Tests for any character that is alphanumeric (<code>isalpha()</code> or <code>isdigit()</code> is true).
Declared In	<code>posix/ctype.h</code>
Prototype	<code>int isalnum (int <i>character</i>)</code>
Parameters	<code>→ <i>character</i></code> The 7-bit ASCII character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the <code>TxtCharIsAlNum()</code> macro; see Exploring Palm OS: Text and Localization .
See Also	isalpha() , isdigit()

isalpha Function

- Purpose** Tests for any character that is a letter in the alphabet (`islower()` or `isupper()` is true).
- Declared In** `posix/ctype.h`
- Prototype** `int isalpha (int character)`
- Parameters** → *character*
The 7-bit ASCII character being evaluated.
- Returns** Returns a non-zero value if the character tests true; zero (0) if the character tests false.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the `TxtCharIsAlpha()` macro; see [Exploring Palm OS: Text and Localization](#).
- See Also** [islower\(\)](#), [isupper\(\)](#)

isblank Function

- Purpose** Tests for any character that is a standard blank-space character.
- Declared In** `posix/ctype.h`
- Prototype** `int isblank (int character)`
- Parameters** → *character*
The 7-bit ASCII character being evaluated.
- Returns** Returns a non-zero value if the character tests true; zero (0) if the character tests false.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use. It only works for 7-bit ASCII.
- See Also** [isspace\(\)](#)

iscntrl Function

Purpose	Tests for any control character.
Declared In	posix/ctype.h
Prototype	int iscntrl (int <i>character</i>)
Parameters	→ <i>character</i> The 7-bit ASCII character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the <code>TxtCharIsCntrl()</code> macro; see Exploring Palm OS: Text and Localization .

isdigit Function

Purpose	Tests for any decimal-digit character.
Declared In	posix/ctype.h
Prototype	int isdigit (int <i>character</i>)
Parameters	→ <i>character</i> The 7-bit ASCII character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the <code>TxtCharIsDigit()</code> macro; see Exploring Palm OS: Text and Localization .

isgraph Function

Purpose	Tests for any printing character except space (' ').
Declared In	posix/ctype.h
Prototype	int isgraph (int <i>character</i>)
Parameters	→ <i>character</i> The 7-bit ASCII character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the <code>TxtCharIsGraph ()</code> macro; see Exploring Palm OS: Text and Localization .

islower Function

Purpose	Tests for any character that is a lowercase letter.
Declared In	posix/ctype.h
Prototype	int islower (int <i>character</i>)
Parameters	→ <i>character</i> The 7-bit ASCII character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the <code>TxtCharIsLower ()</code> macro; see Exploring Palm OS: Text and Localization .
See Also	isupper ()

isprint Function

Purpose	Tests for any printing character including space (' ').
Declared In	posix/ctype.h
Prototype	int isprint (int <i>character</i>)
Parameters	→ <i>character</i> The 7-bit ASCII character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the <code>TxtCharIsPrint()</code> macro; see Exploring Palm OS: Text and Localization .

ispunct Function

Purpose	Tests for any printing character that is a punctuation character.
Declared In	posix/ctype.h
Prototype	int ispunct (int <i>character</i>)
Parameters	→ <i>character</i> The 7-bit ASCII character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the <code>TxtCharIsPunct()</code> macro; see Exploring Palm OS: Text and Localization .

isspace Function

- Purpose** Tests for any printing character that is a standard white-space character.
- Declared In** `posix/ctype.h`
- Prototype** `int isspace (int character)`
- Parameters** `→ character`
The 7-bit ASCII character being evaluated.
- Returns** Returns a non-zero value if the character tests true; zero (0) if the character tests false.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the `TxtCharIsSpace ()` macro; see [Exploring Palm OS: Text and Localization](#).

isupper Function

- Purpose** Tests for any printing character that is an uppercase letter.
- Declared In** `posix/ctype.h`
- Prototype** `int isupper (int character)`
- Parameters** `→ character`
The 7-bit ASCII character being evaluated.
- Returns** Returns a non-zero value if the character tests true; zero (0) if the character tests false.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the `TxtCharIsUpper ()` macro; see [Exploring Palm OS: Text and Localization](#).
- See Also** [islower\(\)](#)

isxdigit Function

Purpose	Tests for any hexadecimal-digit character.
Declared In	<code>posix/ctype.h</code>
Prototype	<code>int isalnum (int <i>character</i>)</code>
Parameters	<code>→ <i>character</i></code> The 7-bit ASCII character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. It only works for 7-bit ASCII. The Palm OS equivalent of this function is the <code>TxtCharIsHex ()</code> macro; see Exploring Palm OS: Text and Localization .

tolower Function

Purpose	Converts an uppercase letter to a corresponding lowercase letter.
Declared In	<code>posix/ctype.h</code>
Prototype	<code>int tolower (int <i>character</i>)</code>
Parameters	<code>→ <i>character</i></code> The character being evaluated.
Returns	Returns a non-zero value if the character tests true; zero (0) if the character tests false.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use. The Palm OS equivalents of this function are the <code>StrToLower ()</code> function and <code>TxtTransliterate ()</code> function; see Exploring Palm OS: Text and Localization .
See Also	<code>toupper ()</code>

toupper Function

- Purpose** Converts a lowercase letter to a corresponding uppercase letter.
- Declared In** `posix/ctype.h`
- Prototype** `int toupper (int character)`
- Parameters** `→ character`
The character being evaluated.
- Returns** Returns a non-zero value if the character tests true; zero (0) if the character tests false.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use. The Palm OS equivalent of this function is the `TxtTransliterate()` function; see [Exploring Palm OS: Text and Localization](#).
- See Also** [tolower\(\)](#)

errno.h

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

Global Variables

errno Variable

Purpose	Global error code variable.
Declared In	<code>posix/errno.h</code>
Prototype	<code>extern int errno</code>
Comments	<p>The <code>errno</code> variable is used by many functions to return error values. The value of <code>errno</code> is defined only after a call to a function for which it is explicitly stated to be set and until it is changed by the next function call. The value of <code>errno</code> should only be examined when it is indicated to be valid by a function's return value. Programs should obtain the definition of <code>errno</code> by the inclusion of <code><errno.h></code>. It is unspecified whether <code>errno</code> is a macro or an identifier declared with external linkage.</p> <p>The <code>errno</code> variable has a value of zero (0) at the beginning. If an error occurs, then this variable is given the value of the error number. In some cases, the behavior of the math library with regard to <code>errno</code> is implementation defined.</p> <p>Nothing in the <code><errno.h></code> header is specific to Palm OS. The specific numeric values associated with the error names are not portable and should be treated as opaque by applications.</p>
See Also	perror() , strerror()

errno.h

errno Variable

fcntl.h

The `<fcntl.h>` header defines several functions useful for porting code from Unix. These functions are *not* part of the ANSI C standard.

Functions and Macros

fcntl Function

Purpose	Manipulates a file descriptor with a specified command.
Declared In	<code>posix/fcntl.h</code>
Prototype	<code>extern int fcntl (int <i>fd</i>, int <i>op</i>, ...)</code>
Parameters	<p>→ <i>fd</i> The file descriptor.</p> <p>→ <i>op</i> The command.</p>
Returns	Returns the file descriptor for the created file upon successful completion. Otherwise, -1 is returned and the global variable <code>errno</code> is set to indicate the error.
Compatibility	This function is <i>not</i> in the C99 specification.

open Function

Purpose	Opens a file and returns its ID.
Declared In	<code>posix/fcntl.h</code>
Prototype	<code>extern int open (const char *<i>pathname</i>, int <i>oflags</i>, ...)</code>
Parameters	<p>→ <i>pathname</i> The filename to open for reading and/or writing.</p>

fcntl.h

open

→ *oflags*

The open mode. The open flags (O_* symbols such as O_RDONLY) are defined in the <fcntl.h> header.

Returns Returns the file ID specified as an integer value.

Compatibility This function is *not* in the C99 specification.

in.h

The `<in.h>` header defines functions useful for converting between Internet host and network addresses.

Structures and Types

sockaddr_in Struct

Purpose	Defines a structure used to store Internet addresses.								
Declared In	<code>posix/netinet/in.h</code>								
Prototype	<pre>struct sockaddr_in { sa_family_t sin_family; in_port_t sin_port; struct in_addr sin_addr; uint8_t sin_zero[8]; }</pre>								
Fields	<table><tr><td><code>sin_family</code></td><td><code>AF_INET</code>.</td></tr><tr><td><code>sin_port</code></td><td>The port number.</td></tr><tr><td><code>sin_addr</code></td><td>The IP address.</td></tr><tr><td><code>sin_zero</code></td><td>The address value.</td></tr></table>	<code>sin_family</code>	<code>AF_INET</code> .	<code>sin_port</code>	The port number.	<code>sin_addr</code>	The IP address.	<code>sin_zero</code>	The address value.
<code>sin_family</code>	<code>AF_INET</code> .								
<code>sin_port</code>	The port number.								
<code>sin_addr</code>	The IP address.								
<code>sin_zero</code>	The address value.								

Functions and Macros

htonl Function

- Purpose** Converts 32-bit values between host byte order and network byte order.
- Declared In** `posix/netinet/in.h`
- Prototype** `uint32_t htonl (uint32_t host32)`
- Parameters** `→ host32`
The value being converted.
- Returns** Returns an unsigned integer.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [gethostbyname\(\)](#), [getservent\(\)](#)

htons Function

- Purpose** Converts 16-bit values between host byte order and network byte order.
- Declared In** `posix/netinet/in.h`
- Prototype** `uint16_t htons (uint16_t host16)`
- Parameters** `→ host16`
The value being converted.
- Returns** Returns an unsigned short integer.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [gethostbyname\(\)](#), [getservent\(\)](#)

ntohl Function

- Purpose** Converts 32-bit values between network byte order and host byte order.
- Declared In** posix/netinet/in.h
- Prototype** uint32_t ntohl (uint32_t net32)
- Parameters** → *net32*
The value being converted.
- Returns** Returns an unsigned integer.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [gethostbyname\(\)](#), [getservent\(\)](#)

ntohs Function

- Purpose** Converts 16-bit values between network byte order and host byte order.
- Declared In** posix/netinet/in.h
- Prototype** uint16_t ntohs (uint16_t net16)
- Parameters** → *net16*
The value being converted.
- Returns** Returns an unsigned short integer.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [gethostbyname\(\)](#), [getservent\(\)](#)

in.h
ntohs

inet.h

The `<inet.h>` header defines several functions useful for Internet address manipulation.

Functions and Macros

inet_addr Function

Purpose	Interprets the specified character string and returns a number suitable for use as an Internet address.
Declared In	<code>posix/arpa/inet.h</code>
Prototype	<code>in_addr_t inet_addr (const char *cp)</code>
Parameters	<code>→ cp</code> A character string.
Returns	Returns a number suitable for use as an Internet address.
Comments	This is a standard network function.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	inet_network()

inet_aton Function

Purpose	Interprets the specified character string as an Internet address, placing the address into the structure provided.
Declared In	<code>posix/arpa/inet.h</code>
Prototype	<code>int inet_aton (const char *cp, struct in_addr *addr)</code>
Parameters	<code>→ cp</code> A character string.

inet.h

inet_lnaof

→ *addr*

An Internet address.

Returns Returns 1 if the string was successfully interpreted, or zero (0) if the string is invalid.

Comments This is a non-standard network function.

Compatibility This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

inet_lnaof Function

Purpose Breaks apart the specified Internet host address and returns the local network address part (in host order).

Declared In `posix/arpa/inet.h`

Prototype `in_addr_t inet_lnaof (struct in_addr in)`

Parameters → *in*

An Internet address.

Returns Returns the local network address (in host order).

Comments This is a non-standard network function.

Compatibility This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

See Also [inet_netof\(\)](#)

inet_makeaddr Function

Purpose Takes an Internet network number and a local network address (both in host order) and constructs an Internet address from it.

Declared In `posix/arpa/inet.h`

Prototype `struct in_addr inet_makeaddr (int net, int lna)`

Parameters → *net*

An Internet network number.

→ *lna*

A local network address.

- Returns** Returns an Internet address.
- Comments** This is a non-standard network function.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

inet_netof Function

- Purpose** Breaks apart the specified Internet host address and returns the network number part (in host order).
- Declared In** `posix/arpa/inet.h`
- Prototype** `in_addr_t inet_netof (struct in_addr in)`
- Parameters** `→ in`
An Internet address.
- Returns** Returns the network number (in host order).
- Comments** This is a non-standard network function.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).
- See Also** [inet_lnaof\(\)](#)

inet_network Function

- Purpose** Interprets the specified character string and returns a number suitable for use as an Internet network number.
- Declared In** `posix/arpa/inet.h`
- Prototype** `in_addr_t inet_network (const char *cp)`
- Parameters** `→ cp`
A character string.
- Returns** Returns a number suitable for use as an Internet network number.
- Comments** This is a non-standard network function.
- Compatibility** This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

See Also [inet_addr\(\)](#)

inet_ntoa Function

Purpose Takes an Internet address and returns an ASCII string representing the address.

Declared In `posix/arpa/inet.h`

Prototype `const char *inet_ntoa (struct in_addr in)`

Parameters `→ in`
An Internet address.

Returns Returns a pointer to an ASCII string representing the address.

Comments This is a standard network function.

Compatibility This function is *not* in the C99 specification.

inet_ntop Function

Purpose Converts a network format address to presentation format.

Declared In `posix/arpa/inet.h`

Prototype `const char *inet_ntop (int af, const void *src,
char *dst, size_t size)`

Parameters `→ af`
The address family.
`→ src`
The source buffer.
`→ dst`
The destination buffer.
`→ size`
The size of the destination buffer.

Returns Returns a pointer to the destination buffer. Otherwise, NULL is returned if a system error occurs and the global variable `errno` is set to indicate the error.

Comments This is a standard network function.

Compatibility This function is *not* in the C99 specification.

inet_pton Function

Purpose Converts a presentation format address to network format.

Declared In `posix/arpa/inet.h`

Prototype `int inet_pton (int af, const char *src,
void *dst)`

Parameters → *af*

The address family.

→ *src*

The printable form as specified in a character string.

→ *dst*

The destination string.

Returns Returns 1 if the address was valid for the specified address family, or zero (0) if the address was not parseable in the specified address family, or -1 if some system error occurred (in which case the global variable `errno` is set to indicate the error).

Comments This is a standard network function.

Compatibility This function is *not* in the C99 specification.

inet.h

inet_pton

ioctl.h

The `<ioctl.h>` header defines a function to manipulate the underlying device parameters of special files.

Functions and Macros

ioctl Function

Purpose	Performs a variety of device-specific control functions on device special files. This function is supported for the following special file types: sockets, console devices, and communication port devices.
Declared In	<code>posix/sys/ioctl.h</code>
Prototype	<pre>int ioctl (int <i>d</i>, unsigned long <i>request</i>, void *<i>argp</i>)</pre>
Parameters	<p>→ <i>d</i> An open file descriptor.</p> <p>→ <i>request</i> An <code>ioctl</code> request.</p> <p>→ <i>argp</i> A pointer to whatever call-specific data is needed by the specific operation being performed. Every <code>ioctl()</code> request has its own requirements.</p>
Returns	Returns -1 if some system error occurred (in which case the global variable <code>errno</code> is set to indicate the error).
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).

ioctl.h

ioctl

iso646.h

The `<iso646.h>` header defines several constants that expand to the corresponding tokens, useful for programming in ISO 646 variant character sets.

Operators

Purpose	Defines constants that expand to the corresponding tokens.
Declared In	<code>posix/iso646.h</code>
Constants	<pre> #define and && The operator &&. #define and_eq &= The operator &=. #define bitand & The operator &. #define bitor The operator . #define compl ~ The operator ~. #define not ! The operator !. #define not_eq != The operator !=. #define or The operator . #define or_eq = The operator =. #define xor ^ The operator ^. #define xor_eq ^= The operator ^=.</pre>

iso646.h

Operators

locale.h

The `<locale.h>` header support in `libc` has not been hooked up with Palm OS and thus should not be used. The macros and functions defined in this header do not work as expected and should be avoided.

math.h

The `<math.h>` header defines several mathematical functions.

This header is new with Palm OS Protein. It is a broad subset of section 7.12 of the C language standard ANSI/ISO/IEC 9899:1999.

MathLib is part of SystemLib. To use MathLib, simply include the `<math.h>` header in your source files.

Supported features

The Palm OS Protein C/C++ Compiler supports the use of infinity and NaN (not-a-number) values.

The following C99 macros are supported in `<math.h>`:

- `FLT_EVAL`
- `FP_ILOGBNAN`
- `FP_ILOGB0`
- `FP_INFINITE`
- `FP_NAN`
- `FP_NORMAL`
- `FP_SUBNORMAL`
- `FP_ZERO`
- `HUGE_VAL`
- `HUGE_VALF`
- `HUGE_VALL`
- `INFINITY`
- `MATH_ERREXCEPT`
- `math_errhandling`
- `MATH_ERRNO`
- `NAN`

Differences from the C99 specification

- All of `<math.h>` as specified in the C language standard ANSI/ISO/IEC 9899:1990 is provided as well as most of the extensions specified in 1999 standard. Parts of `<math.h>` that are not supported are listed under the line:

```
#ifndef __USE_C99_EXTENSIONS__
```

Functions in this section are preprocessed out by default and are not tested or supplied by PalmSource.

- Parallel sets of functions for `float` and `long double` arguments types are defined only for 1989 ANSI C functions.

Constraints

- Existing 68K applications must continue to supply the 68K MathLib if required by the application.
- There are some cases in this version of MathLib where the global variable `errno` does not get set when it should.
- The `float` and `long double` overloads as specified in section 26.5 of the ANSI C++ standard are *not* provided.
- The `float` and `long double` counterparts suffixed by “f” and “l” for the functions defined in section 7.12 of the 1989 ANSI C language standard are supported. A few of the `float` counterparts have Palm OS implementations, but most of these simply cast and return the `double` version.
- A handful of single precision counterparts are provided as a high performance alternative to their `double` equivalents. However, there are some additional deviations from the standard that were made to achieve high performance, including:
 - none of the single precision functions set the global variable `errno`.
 - `sqrtf()` flushes denormals to zero (0).
 - `ceilf(-0)` is 0 not `-0` as specified in Annex F.9.6.1 of the ANSI standard.
 - `hypotf()` does not follow the spec for NaNs and infinities.
- The library, `libm.a`, is no longer supported and must be removed from existing projects.

Functions and Macros

abs Function

Purpose	Computes the absolute value of x .
Declared In	posix/math.h
Prototype	double abs (double x) float abs (float x) long double abs (long double x)
Parameters	→ x Value to be evaluated.
Returns	Returns the absolute value of x .
Comments	Chapter 26.5 of the ANSI C++ standard modifies <math.h> by adding the abs () overloads. These overloads are equivalent to the ANSI standard fabs () family of functions.
See Also	fabs ()

acos Function

Purpose	Computes the arc-cosine of x .
Declared In	posix/math.h
Prototype	double acos (double x)
Parameters	→ x Value of type double to be evaluated. The value x must be within the range of -1 to +1 (inclusive).
Returns	Returns the arc-cosine of x , a value within the range of zero (0) to π (inclusive).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	asin () , atan ()

acosf Function

Purpose	Computes the arc-cosine of x .
Declared In	posix/math.h
Prototype	float acosf (float x)
Parameters	→ x Value of type float to be evaluated.
Returns	Returns the arc-cosine of x , a value within the range of zero (0) to π (inclusive).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	asinf() , atanf()

acosh Function

Purpose	Computes the inverse hyperbolic cosine of x .
Declared In	posix/math.h
Prototype	double acosh (double x)
Parameters	→ x Value of type double to be evaluated. The value of x has no range limit.
Returns	Returns the inverse hyperbolic cosine of x , a value without a range limit.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	asinh() , atanh()

acosl Function

Purpose	Computes the arc-cosine of x .
Declared In	posix/math.h
Prototype	long double acosl (long double x)
Parameters	→ x Value of type long double to be evaluated.

- Returns** Returns the arc-cosine of x , a value within the range of zero (0) to π (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [asinl\(\)](#), [atanl\(\)](#)

asin Function

- Purpose** Computes the arc-sine of x .
- Declared In** `posix/math.h`
- Prototype** `double asin (double x)`
- Parameters** $\rightarrow x$
Value of type `double` to be evaluated. The value x must be within the range of -1 to +1 (inclusive).
- Returns** Returns the arc-sine of x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [acos\(\)](#), [atan\(\)](#)

asinf Function

- Purpose** Computes the arc-sine of x .
- Declared In** `posix/math.h`
- Prototype** `float asinf (float x)`
- Parameters** $\rightarrow x$
Value of type `float` to be evaluated.
- Returns** Returns the arc-sine of x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [acosf\(\)](#), [atanf\(\)](#)

asinh Function

- Purpose** Computes the inverse hyperbolic sine of x .
- Declared In** `posix/math.h`
- Prototype** `double asinh (double x)`
- Parameters** $\rightarrow x$
Value of type `double` to be evaluated. The value of x has no range limit.
- Returns** Returns the inverse hyperbolic sine of x , a value without a range limit.
- Compatibility** This function *is* in the C99 specification.
- See Also** [acosh\(\)](#), [atanh\(\)](#)

asinl Function

- Purpose** Computes the arc-sine of x .
- Declared In** `posix/math.h`
- Prototype** `long double asinl (long double x)`
- Parameters** $\rightarrow x$
Value of type `long double` to be evaluated.
- Returns** Returns the arc-sine of x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [acosl\(\)](#), [atanl\(\)](#)

atan Function

- Purpose** Computes the arc-tangent of x .
- Declared In** `posix/math.h`
- Prototype** `double atan (double x)`
- Parameters** $\rightarrow x$
Value of type `double` to be evaluated. The value of x has no range limit.

- Returns** Returns the arc-tangent of x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [acos\(\)](#), [asin\(\)](#), [atan2\(\)](#)

atan2 Function

- Purpose** Computes the arc-tangent of y/x .
- Declared In** `posix/math.h`
- Prototype** `double atan2 (double y , double x)`
- Parameters**
- $\rightarrow y$
Value of type `double` to be evaluated.
 - $\rightarrow x$
Value of type `double` to be evaluated.
- Returns** Returns the arc-tangent of y/x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
- Comments** Both x and y cannot be zero (0).
- Compatibility** This function *is* in the C99 specification.
- See Also** [atan\(\)](#)

atan2f Function

- Purpose** Computes the arc-tangent of y/x .
- Declared In** `posix/math.h`
- Prototype** `float atan2f (float y , float x)`
- Parameters**
- $\rightarrow y$
Value of type `float` to be evaluated.
 - $\rightarrow x$
Value of type `float` to be evaluated.
- Returns** Returns the arc-tangent of y/x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [atanf\(\)](#)

atan2l Function

- Purpose** Computes the arc-tangent of y/x .
- Declared In** `posix/math.h`
- Prototype** `long double atan2l (long double y, long double x)`
- Parameters**
- $\rightarrow y$
Value of type `long double` to be evaluated.
 - $\rightarrow x$
Value of type `long double` to be evaluated.
- Returns** Returns the arc-tangent of y/x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [atanl\(\)](#)

atanf Function

- Purpose** Computes the arc-tangent of x .
- Declared In** `posix/math.h`
- Prototype** `float atanf (float x)`
- Parameters**
- $\rightarrow x$
Value of type `float` to be evaluated.
- Returns** Returns the arc-tangent of x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [acosf\(\)](#), [asinf\(\)](#), [atan2f\(\)](#)

atanh Function

Purpose	Computes the inverse hyperbolic tangent of x .
Declared In	posix/math.h
Prototype	double atanh (double x)
Parameters	→ x Value of type double to be evaluated. The value of x has no range limit.
Returns	Returns the inverse hyperbolic tangent of x , a value without a range limit.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	acosh() , asinh()

atanl Function

Purpose	Computes the arc-tangent of x .
Declared In	posix/math.h
Prototype	long double atanl (long double x)
Parameters	→ x Value of type long double to be evaluated.
Returns	Returns the arc-tangent of x , a value within the range of $-\pi/2$ to $+\pi/2$ (inclusive).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	acosl() , asinl() , atan2l()

cbrt Function

Purpose	Computes the cube root of x .
Declared In	posix/math.h
Prototype	double cbrt (double x)
Parameters	→ x Value of type double to be evaluated.
Returns	Returns the cube root of x .
Compatibility	This function <i>is</i> in the C99 specification.

ceil Function

- Purpose** Computes the smallest integer not less than *x*.
- Declared In** `posix/math.h`
- Prototype** `double ceil (double x)`
- Parameters** $\rightarrow x$
Value of type `double` to be evaluated. The value of *x* has no range limit.
- Returns** Returns the smallest integer not less than *x*, a value without a range limit.
- Compatibility** This function *is* in the C99 specification.
- See Also** [ceilf\(\)](#), [ceil1\(\)](#)

ceilf Function

- Purpose** Computes the smallest integer not less than *x*.
- Declared In** `posix/math.h`
- Prototype** `float ceilf (float x)`
- Parameters** $\rightarrow x$
Value of type `float` to be evaluated. The value of *x* has no range limit.
- Returns** Returns the smallest integer not less than *x*, a value without a range limit.
- Compatibility** This function *is* in the C99 specification.
- See Also** [ceil\(\)](#), [ceil1\(\)](#)

ceil1 Function

- Purpose** Computes the smallest integer not less than *x*.
- Declared In** `posix/math.h`
- Prototype** `long double ceil1 (long double x)`
- Parameters** $\rightarrow x$
Value of type `long double` to be evaluated.

- Returns** Returns the smallest integer not less than *x*, a value without a range limit.
- Compatibility** This function *is* in the C99 specification.
- See Also** [ceil\(\)](#), [ceilf\(\)](#)

copysign Function

- Purpose** Returns *x* with its sign changed to *y*'s.
- Declared In** `posix/math.h`
- Prototype** `double copysign (double x, double y)`
- Parameters**
- *x*
Value of type `double` to be changed.
 - *y*
Value of type `double` to be evaluated.
- Returns** Returns the value of *x* with its sign changed to *y*'s.
- Compatibility** This function *is* in the C99 specification.

cos Function

- Purpose** Computes the cosine of *x*.
- Declared In** `posix/math.h`
- Prototype** `double cos (double x)`
- Parameters**
- *x*
Value of type `double` to be evaluated. The value of *x* has no range limit.
- Returns** Returns the cosine of *x*, a value within the range of -1 to +1 (inclusive).
- Compatibility** This function *is* in the C99 specification.
- See Also** [sin\(\)](#), [tan\(\)](#)

cosf Function

Purpose	Computes the cosine of x .
Declared In	posix/math.h
Prototype	float cosf (float x)
Parameters	→ x Value of type float to be evaluated.
Returns	Returns the cosine of x , a value within the range of -1 to +1 (inclusive).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	sinf() , tanf()

cosh Function

Purpose	Computes the hyperbolic cosine of x .
Declared In	posix/math.h
Prototype	double cosh (double x)
Parameters	→ x Value of type double to be evaluated. The value of x has no range limit.
Returns	Returns the hyperbolic cosine of x , a value without a range limit.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	sinh() , tanh()

coshf Function

Purpose	Computes the hyperbolic cosine of x .
Declared In	posix/math.h
Prototype	float coshf (float x)
Parameters	→ x Value of type float to be evaluated.
Returns	Returns the hyperbolic cosine of x , a value without a range limit.

Compatibility This function *is* in the C99 specification.

See Also [sinhf\(\)](#), [tanhf\(\)](#)

coshl Function

Purpose Computes the hyperbolic cosine of x .

Declared In `posix/math.h`

Prototype `long double coshl (long double x)`

Parameters $\rightarrow x$
Value of type `long double` to be evaluated.

Returns Returns the hyperbolic cosine of x , a value without a range limit.

Compatibility This function *is* in the C99 specification.

See Also [sinhl\(\)](#), [tanhl\(\)](#)

cosl Function

Purpose Computes the cosine of x .

Declared In `posix/math.h`

Prototype `long double cosl (long double x)`

Parameters $\rightarrow x$
Value of type `long double` to be evaluated.

Returns Returns the cosine of x , a value within the range of -1 to +1 (inclusive).

Compatibility This function *is* in the C99 specification.

See Also [asinl\(\)](#), [atanl\(\)](#)

exp Function

Purpose Computes the exponential of x .

Declared In `posix/math.h`

Prototype `double exp (double x)`

- Parameters** → *x*
Value of type `double` to be evaluated.
- Returns** Returns the exponential of *x*.
- Compatibility** This function *is* in the C99 specification.
- See Also** [expf\(\)](#), [expl\(\)](#), [expm1\(\)](#)

expf Function

- Purpose** Computes the exponential of *x*.
- Declared In** `posix/math.h`
- Prototype** `float expf (float x)`
- Parameters** → *x*
Value of type `float` to be evaluated.
- Returns** Returns the exponential of *x*.
- Comments** The global variable `errno` does not get set when underflow occurs.
- Compatibility** This function *is* in the C99 specification.
- See Also** [exp\(\)](#), [expl\(\)](#)

expl Function

- Purpose** Computes the exponential of *x*.
- Declared In** `posix/math.h`
- Prototype** `long double expl (long double x)`
- Parameters** → *x*
Value of type `long double` to be evaluated.
- Returns** Returns the exponential of *x*.
- Compatibility** This function *is* in the C99 specification.
- See Also** [exp\(\)](#), [expf\(\)](#)

expm1 Function

Purpose	Computes the value of $\exp(x)-1$ accurately even for tiny argument x .
Declared In	posix/math.h
Prototype	double expm1 (double x)
Parameters	$\rightarrow x$ Value of type double to be evaluated.
Returns	Returns the value of $\exp(x)-1$.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	exp()

fabs Function

Purpose	Computes the absolute value of x .
Declared In	posix/math.h
Prototype	double fabs (double x)
Parameters	$\rightarrow x$ Value of type double to be evaluated.
Returns	Returns the absolute value of x .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	fabsf() , fabsl()

fabsf Function

Purpose	Computes the absolute value of x .
Declared In	posix/math.h
Prototype	float fabsf (float x)
Parameters	$\rightarrow x$ Value of type float to be evaluated.
Returns	Returns the absolute value of x .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	fabs() , fabsl()

fabsl Function

Purpose	Computes the absolute value of <i>x</i> .
Declared In	posix/math.h
Prototype	long double fabsl (long double <i>x</i>)
Parameters	→ <i>x</i> Value of type long double to be evaluated.
Returns	Returns the absolute value of <i>x</i> .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	fabs() , fabsf()

floor Function

Purpose	Computes the largest integer not greater than <i>x</i> .
Declared In	posix/math.h
Prototype	double floor (double <i>x</i>)
Parameters	→ <i>x</i> Value of type double to be evaluated.
Returns	Returns the largest integer not greater than <i>x</i> .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	floorf() , floorl()

floorf Function

Purpose	Computes the largest integer not greater than <i>x</i> .
Declared In	posix/math.h
Prototype	float floorf (float <i>x</i>)
Parameters	→ <i>x</i> Value of type float to be evaluated.
Returns	Returns the largest integer not greater than <i>x</i> .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	floor() , floorl()

floorl Function

Purpose	Computes the largest integer not greater than x .
Declared In	<code>posix/math.h</code>
Prototype	<code>long double floorl (long double x)</code>
Parameters	$\rightarrow x$ Value of type <code>long double</code> to be evaluated.
Returns	Returns the largest integer not greater than x .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	floor() , floorf()

fmod Function

Purpose	Computes the floating-point remainder of x/y with same sign as x (if y is non-zero). If y is zero (0), the result is implementation-defined.
Declared In	<code>posix/math.h</code>
Prototype	<code>double fmod (double x, double y)</code>
Parameters	$\rightarrow x$ Value of type <code>double</code> to be evaluated. $\rightarrow y$ Value of type <code>double</code> to be evaluated.
Returns	Returns the floating-point remainder, unless y is zero (0).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	fmodf() , fmodl()

fmodf Function

Purpose	Computes the floating-point remainder of x/y with same sign as x (if y is non-zero). If y is zero (0), the result is implementation-defined.
Declared In	<code>posix/math.h</code>
Prototype	<code>float fmodf (float x, float y)</code>
Parameters	$\rightarrow x$ Value of type <code>float</code> to be evaluated. $\rightarrow y$ Value of type <code>float</code> to be evaluated.
Returns	Returns the floating-point remainder, unless y is zero (0).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	fmod() , fmodl()

fmodl Function

Purpose	Computes the floating-point remainder of x/y with same sign as x (if y is non-zero). If y is zero (0), the result is implementation-defined.
Declared In	<code>posix/math.h</code>
Prototype	<code>long double fmodl (long double x, long double y)</code>
Parameters	$\rightarrow x$ Value of type <code>long double</code> to be evaluated. $\rightarrow y$ Value of type <code>long double</code> to be evaluated.
Returns	Returns the floating-point remainder, unless y is zero (0).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	fmod() , fmodf()

fpclassify Macro

- Purpose** Classifies its argument value as NaN, infinite, normal, subnormal, zero, or into another implementation-defined category.
- Declared In** `posix/math.h`
- Prototype** `#define fpclassify (real-floating x)`
- Parameters** $\rightarrow x$
Value of type `real floating` to be evaluated.
- Returns** Returns the value of the number classification macro appropriate to the value of its argument.
- Compatibility** This function *is* in the C99 specification; see classification macros section 7.12.3 in the standards document.
- See Also** [signbit\(\)](#), [isinf\(\)](#), [isnan\(\)](#), [isnormal\(\)](#), [signbit\(\)](#)

frexp Function

- Purpose** Breaks up the floating-point number `x` into a mantissa and exponent.
- Declared In** `posix/math.h`
- Prototype** `double frexp (double x, int *exp)`
- Parameters** $\rightarrow x$
Value of type `double` to be evaluated.
 $\rightarrow exp$
Value of type `int` to be evaluated. The mantissa and the integer pointed to by `exp` is the exponent.
- Returns** Returns the value of $x = \text{mantissa} * 2^{\text{exp}}$, unless `x` is zero (0). If `x` is zero, both `*exp` and the result are zero.
- Compatibility** This function *is* in the C99 specification.
- See Also** [frexpf\(\)](#), [frexpl\(\)](#)

frexpf Function

- Purpose** Breaks up the floating-point number *x* into a mantissa and exponent.
- Declared In** `posix/math.h`
- Prototype** `float frexpf (float value, int *exp)`
- Parameters**
- *value*
Value of type `float` to be evaluated.
 - *exp*
Value of type `int` to be evaluated. The mantissa and the integer pointed to by *exp* is the exponent.
- Returns** Returns the value of $x = \text{mantissa} * 2^{\text{exp}}$, unless *x* is zero (0). If *x* is zero, both **exp* and the result are zero.
- Compatibility** This function *is* in the C99 specification.
- See Also** [frexp\(\)](#), [frexpl\(\)](#)

frexpl Function

- Purpose** Breaks up the floating-point number *x* into a mantissa and exponent.
- Declared In** `posix/math.h`
- Prototype** `long double frexpl (long double value, int *exp)`
- Parameters**
- *value*
Value of type `long double` to be evaluated.
 - *exp*
Value of type `int` to be evaluated. The mantissa and the integer pointed to by *exp* is the exponent.
- Returns** Returns the value of $x = \text{mantissa} * 2^{\text{exp}}$, unless *x* is zero (0). If *x* is zero, both **exp* and the result are zero.
- Compatibility** This function *is* in the C99 specification.
- See Also** [frexp\(\)](#), [frexpf\(\)](#)

hypot Function

- Purpose** Computes the `sqrt(x*x+y*y)` in such a way that underflow does not happen, and overflow occurs only if the final result deserves it.
- Declared In** `posix/math.h`
- Prototype** `double hypot (double x, double y)`
- Parameters**
- `x`
Value of type `double` to be evaluated.
 - `y`
Value of type `double` to be evaluated.
- Returns** Returns the value of `sqrt(x*x+y*y)`.
- Compatibility** This function *is* in the C99 specification.
- See Also** [hypotf\(\)](#), [hypotl\(\)](#)

hypotf Function

- Purpose** Computes the `sqrt(x*x+y*y)` in such a way that underflow does not happen, and overflow occurs only if the final result deserves it.
- Declared In** `posix/math.h`
- Prototype** `float hypot (float x, float y)`
- Parameters**
- `x`
Value of type `float` to be evaluated.
 - `y`
Value of type `float` to be evaluated.
- Returns** Returns the value of `sqrt(x*x+y*y)`.
- Compatibility** This function *is* in the C99 specification.
- See Also** [hypot\(\)](#), [hypotl\(\)](#)

hypotl Function

Purpose	Computes the <code>sqrt(x*x+y*y)</code> in such a way that underflow does not happen, and overflow occurs only if the final result deserves it.
Declared In	<code>posix/math.h</code>
Prototype	<code>long double hypotl (long double x, long double y)</code>
Parameters	<code>→ x</code> Value of type <code>long double</code> to be evaluated. <code>→ y</code> Value of type <code>long double</code> to be evaluated.
Returns	Returns the value of <code>sqrt(x*x+y*y)</code> .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	hypot() , hypotf()

ilogb Function

Purpose	Computes <i>x</i> 's exponent <i>n</i> .
Declared In	<code>posix/math.h</code>
Prototype	<code>int ilogb (double x)</code>
Parameters	<code>→ x</code> Value of type <code>double</code> to be evaluated.
Returns	Returns the value of <i>x</i> 's exponent <i>n</i> , in integer format.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	logb()

isfinite Macro

Purpose	Tests for finite value (zero, subnormal, or normal, and not infinite or NaN).
Declared In	<code>posix/math.h</code>
Prototype	<code>#define isfinite (real-floating x)</code>
Parameters	<code>→ x</code> Value of type <code>real floating</code> to be evaluated.

- Returns** Returns a non-zero value if and only if *x* has a finite value.
- Compatibility** This function *is* in the C99 specification; see classification macros section 7.12.3 in the standards document.
- See Also** [fpclassify\(\)](#), [isinf\(\)](#), [isnan\(\)](#), [isnormal\(\)](#), [signbit\(\)](#)

isinf Macro

- Purpose** Tests for infinity (positive or negative).
- Declared In** `posix/math.h`
- Prototype** `#define isinf (real-floating x)`
- Parameters** `→ x`
Value of type `real floating` to be evaluated.
- Returns** Returns a non-zero value if and only if *x* has an infinite value.
- Compatibility** This function *is* in the C99 specification; see classification macros section 7.12.3 in the standards document.
- See Also** [fpclassify\(\)](#), [signbit\(\)](#), [isnan\(\)](#), [isnormal\(\)](#), [signbit\(\)](#)

isnan Macro

- Purpose** Tests for a NaN (not a number).
- Declared In** `posix/math.h`
- Prototype** `#define isnan (real-floating x)`
- Parameters** `→ x`
Value of type `real floating` to be evaluated.
- Returns** Returns a non-zero value if and only if *x* has a NaN value.
- Compatibility** This function *is* in the C99 specification; see classification macros section 7.12.3 in the standards document.
- See Also** [fpclassify\(\)](#), [signbit\(\)](#), [isinf\(\)](#), [isnormal\(\)](#), [signbit\(\)](#)

isnormal Macro

- Purpose** Tests for a normal value (neither zero, subnormal, infinite, nor NaN).
- Declared In** `posix/math.h`
- Prototype** `#define isnormal (real-floating x)`
- Parameters** $\rightarrow x$
Value of type `real floating` to be evaluated.
- Returns** Returns a non-zero value if and only if x has a normal value.
- Compatibility** This function *is* in the C99 specification; see classification macros section 7.12.3 in the standards document.
- See Also** [fpclassify\(\)](#), [signbit\(\)](#), [isinf\(\)](#), [isnan\(\)](#), [signbit\(\)](#)

ldexp Function

- Purpose** Computes x multiplied by 2 to the power n .
- Declared In** `posix/math.h`
- Prototype** `double ldexp (double x, int n)`
- Parameters** $\rightarrow x$
Value of type `double` to be evaluated.
- $\rightarrow n$
Value of type `int` to be evaluated.
- Returns** Returns the value of x multiplied by 2 to the power n .
- Compatibility** This function *is* in the C99 specification.
- See Also** [ldexpf\(\)](#), [ldexpl\(\)](#)

ldexpf Function

- Purpose** Computes x multiplied by 2 to the power n .
- Declared In** `posix/math.h`
- Prototype** `float ldexpf (float x, int exp)`
- Parameters** $\rightarrow x$
Value of type `float` to be evaluated.

→ *exp*
Value of type `int` to be evaluated.

- Returns** Returns the value of *x* multiplied by 2 to the power *exp*.
- Compatibility** This function *is* in the C99 specification.
- See Also** [ldexp\(\)](#), [ldexpl\(\)](#)

ldexpl Function

- Purpose** Computes *x* multiplied by 2 to the power *n*.
- Declared In** `posix/math.h`
- Prototype** `long double ldexpl (long double x, int exp)`
- Parameters**
- *x*
Value of type `long double` to be evaluated.
- *exp*
Value of type `int` to be evaluated.
- Returns** Returns the value of *x* multiplied by 2 to the power *exp*.
- Compatibility** This function *is* in the C99 specification.
- See Also** [ldexp\(\)](#), [ldexpf\(\)](#)

log Function

- Purpose** Computes the natural logarithm of *x*.
- Declared In** `posix/math.h`
- Prototype** `double log (double x)`
- Parameters**
- *x*
Value of type `double` to be evaluated.
- Returns** Returns the natural logarithm of *x*.
- Compatibility** This function *is* in the C99 specification.
- See Also** [logf\(\)](#), [logl\(\)](#)

log10 Function

Purpose	Computes the base-10 logarithm of x .
Declared In	posix/math.h
Prototype	double log10 (double x)
Parameters	→ x Value of type double to be evaluated.
Returns	Returns the base-10 logarithm of x .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	log10f() , log10l()

log10f Function

Purpose	Computes the base-10 logarithm of x .
Declared In	posix/math.h
Prototype	float log10f (float x)
Parameters	→ x Value of type float to be evaluated.
Returns	Returns the base-10 logarithm of x .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	log10() , log10l()

log10l Function

Purpose	Computes the base-10 logarithm of x .
Declared In	posix/math.h
Prototype	long double log10l (long double x)
Parameters	→ x Value of type long double to be evaluated.
Returns	Returns the base-10 logarithm of x .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	log10() , log10f()

log1p Function

Purpose	Computes the value of $\log(1+x)$ accurately even for tiny argument x .
Declared In	posix/math.h
Prototype	double log1p (double x)
Parameters	→ x Value of type double to be evaluated.
Returns	Returns the value of $\log(1+x)$.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	log()

logb Function

Purpose	Computes x 's exponent n , a signed integer converted to double-precision floating-point.
Declared In	posix/math.h
Prototype	double logb (double x)
Parameters	→ x Value of type double to be evaluated.
Returns	Returns the value of x 's exponent n , in double format.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	ilogb()

logf Function

Purpose	Computes the natural logarithm of x .
Declared In	posix/math.h
Prototype	float logf (float x)
Parameters	→ x Value of type float to be evaluated.
Returns	Returns the natural logarithm of x .

Compatibility This function *is* in the C99 specification.

See Also [log\(\)](#), [logl\(\)](#)

logl Function

Purpose Computes the natural logarithm of *x*.

Declared In `posix/math.h`

Prototype `long double logl (long double x)`

Parameters `→ x`
Value of type `long double` to be evaluated.

Returns Returns the natural logarithm of *x*.

Compatibility This function *is* in the C99 specification.

See Also [log\(\)](#), [logf\(\)](#)

modf Function

Purpose Computes the fractional part and assigns to *ip* the integral part of *x*, both with same sign as *x*.

Declared In `posix/math.h`

Prototype `double modf (double x, double *ip)`

Parameters `→ x`
Value of type `double` to be evaluated.
`→ ip`
Value of type `double` to be evaluated.

Returns Returns the signed fractional part.

Compatibility This function *is* in the C99 specification.

See Also [modff\(\)](#), [modfl\(\)](#)

modff Function

- Purpose** Computes the fractional part and assigns to *ip* the integral part of *x*, both with same sign as *x*.
- Declared In** `posix/math.h`
- Prototype** `float modff (float value, float *iptr)`
- Parameters**
- *value*
Value of type `float` to be evaluated.
 - *iptr*
Value of type `float` to be evaluated.
- Returns** Returns the signed fractional part.
- Compatibility** This function *is* in the C99 specification.
- See Also** [modf\(\)](#), [modfl\(\)](#)

modfl Function

- Purpose** Computes the fractional part and assigns to *ip* the integral part of *x*, both with same sign as *x*.
- Declared In** `posix/math.h`
- Prototype** `long double modfl (long double value,
long double *iptr)`
- Parameters**
- *value*
Value of type `long double` to be evaluated.
 - *iptr*
Value of type `long double` to be evaluated.
- Returns** Returns the signed fractional part.
- Compatibility** This function *is* in the C99 specification.
- See Also** [modf\(\)](#), [modff\(\)](#)

nextafter Function

Purpose	Computes the next machine representable number from <i>x</i> in direction <i>y</i> .
Declared In	posix/math.h
Prototype	double nextafter (double <i>x</i> , double <i>y</i>)
Parameters	→ <i>x</i> Value of type double to be evaluated. → <i>y</i> Value of type double to be evaluated.
Returns	Returns the next machine representable number from <i>x</i> in direction <i>y</i> .
Compatibility	This function <i>is</i> in the C99 specification.

pow Function

Purpose	Computes <i>x</i> raised to power <i>y</i> .
Declared In	posix/math.h
Prototype	double pow (double <i>x</i> , double <i>y</i>)
Parameters	→ <i>x</i> Value of type double to be evaluated. → <i>y</i> Value of type double to be evaluated.
Returns	Returns the value of <i>x</i> raised to power <i>y</i> .
Comments	pow(infinity, infinity) sets the global variable errno to ERANGE.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	powf() , powl()

powf Function

Purpose	Computes x raised to power y .
Declared In	<code>posix/math.h</code>
Prototype	<code>float powf (float x, float y)</code>
Parameters	$\rightarrow x$ Value of type <code>float</code> to be evaluated. $\rightarrow y$ Value of type <code>float</code> to be evaluated.
Returns	Returns the value of x raised to power y .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	pow() , powl()

powl Function

Purpose	Computes x raised to power y .
Declared In	<code>posix/math.h</code>
Prototype	<code>long double powl (long double x, long double y)</code>
Parameters	$\rightarrow x$ Value of type <code>long double</code> to be evaluated. $\rightarrow y$ Value of type <code>long double</code> to be evaluated.
Returns	Returns the value of x raised to power y .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	pow() , powf()

remainder Function

- Purpose** Computes the remainder $r := x - n * y$ where n is the integer nearest the exact value of x/y .
- Declared In** `posix/math.h`
- Prototype** `double remainder (double x, double y)`
- Parameters**
- $\rightarrow x$
Value of type `double` to be evaluated.
 - $\rightarrow y$
Value of type `double` to be evaluated.
- Returns** Returns the remainder.
- Compatibility** This function *is* in the C99 specification.

rint Function

- Purpose** Rounds x to integral value in floating-point format.
- Declared In** `posix/math.h`
- Prototype** `double rint (double x)`
- Parameters**
- $\rightarrow x$
Value of type `double` to be evaluated.
- Returns** Returns the integral value (represented as a double precision number) nearest to x according to the prevailing rounding mode.
- Compatibility** This function *is* in the C99 specification.

scalbn Function

- Purpose** Computes $x^{(2^{**}n)}$ by exponent manipulation.
- Declared In** `posix/math.h`
- Prototype** `double scalbn (double x, int n)`
- Parameters**
- $\rightarrow x$
Value of type `double` to be evaluated.
 - $\rightarrow n$
Value of type `int` to be evaluated.

Returns Returns the value of $x^{*(2**n)}$.
Compatibility This function *is* in the C99 specification.

signbit Macro

Purpose Tests for a negative sign. (NaNs, zeros, and infinities have a sign bit.)
Declared In `posix/math.h`
Prototype `#define signbit (real-floating x)`
Parameters $\rightarrow x$
Value of type `real floating` to be evaluated.
Returns Returns a non-zero value if and only if the sign of `x` is negative.
Compatibility This function *is* in the C99 specification; see classification macros section 7.12.3 in the standards document.
See Also [fpclassify\(\)](#), [signbit\(\)](#), [isinf\(\)](#), [isnan\(\)](#), [isnormal\(\)](#)

sin Function

Purpose Computes the sine of `x`.
Declared In `posix/math.h`
Prototype `double sin (double x)`
Parameters $\rightarrow x$
Value of type `double` to be evaluated. The value of `x` has no range limit.
Returns Returns the sine of `x`, a value within the range of -1 to +1 (inclusive).
Compatibility This function *is* in the C99 specification.
See Also [cos\(\)](#), [tan\(\)](#)

sinf Function

Purpose	Computes the sine of <i>x</i> .
Declared In	posix/math.h
Prototype	float <code>sinf</code> (float <i>x</i>)
Parameters	→ <i>x</i> Value of type <code>float</code> to be evaluated.
Returns	Returns the sine of <i>x</i> , a value within the range of -1 to +1 (inclusive).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	cosf() , tanf()

sinh Function

Purpose	Computes the hyperbolic sine of <i>x</i> .
Declared In	posix/math.h
Prototype	double <code>sinh</code> (double <i>x</i>)
Parameters	→ <i>x</i> Value of type <code>double</code> to be evaluated. The value of <i>x</i> has no range limit.
Returns	Returns the hyperbolic sine of <i>x</i> , a value without a range limit.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	cosh() , tanh()

sinhf Function

Purpose	Computes the hyperbolic sine of <i>x</i> .
Declared In	posix/math.h
Prototype	float <code>sinhf</code> (float <i>x</i>)
Parameters	→ <i>x</i> Value of type <code>float</code> to be evaluated.
Returns	Returns the hyperbolic sine of <i>x</i> , a value without a range limit.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	coshf() , tanhf()

sinhl Function

Purpose	Computes the hyperbolic sine of <i>x</i> .
Declared In	<code>posix/math.h</code>
Prototype	<code>long double sinhl (long double x)</code>
Parameters	<code>→ x</code> Value of type <code>long double</code> to be evaluated.
Returns	Returns the hyperbolic sine of <i>x</i> , a value without a range limit.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	<code>coshl()</code> , <code>tanh1()</code>

sinl Function

Purpose	Computes the sine of <i>x</i> .
Declared In	<code>posix/math.h</code>
Prototype	<code>long double sinl (long double x)</code>
Parameters	<code>→ x</code> Value of type <code>long double</code> to be evaluated.
Returns	Returns the sine of <i>x</i> , a value within the range of -1 to +1 (inclusive).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	<code>cosl()</code> , <code>tanl()</code>

sqrt Function

Purpose	Computes the non-negative square root of <i>x</i> .
Declared In	<code>posix/math.h</code>
Prototype	<code>double sqrt (double x)</code>
Parameters	<code>→ x</code> Value of type <code>double</code> to be evaluated.
Returns	Returns the non-negative square root of <i>x</i> .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	<code>sqrtf()</code> , <code>sqrtl()</code>

sqrtf Function

Purpose	Computes the non-negative square root of <i>x</i> .
Declared In	posix/math.h
Prototype	float sqrtf (float <i>x</i>)
Parameters	→ <i>x</i> Value of type float to be evaluated.
Returns	Returns the non-negative square root of <i>x</i> .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	sqrt() , sqrtl()

sqrtl Function

Purpose	Computes the non-negative square root of <i>x</i> .
Declared In	posix/math.h
Prototype	long double sqrtl (long double <i>x</i>)
Parameters	→ <i>x</i> Value of type long double to be evaluated.
Returns	Returns the non-negative square root of <i>x</i> .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	sqrt() , sqrtf()

tan Function

Purpose	Computes the tangent of <i>x</i> .
Declared In	posix/math.h
Prototype	double tan (double <i>x</i>)
Parameters	→ <i>x</i> Value of type double to be evaluated. The value of <i>x</i> has no range limit.
Returns	Returns the tangent of <i>x</i> , a value within the range of -1 to +1 (inclusive).

Compatibility This function *is* in the C99 specification.

See Also [cos\(\)](#), [sin\(\)](#)

tanf Function

Purpose Computes the tangent of *x*.

Declared In `posix/math.h`

Prototype `float tanf (float x)`

Parameters $\rightarrow x$
Value of type `float` to be evaluated.

Returns Returns the tangent of *x*, a value within the range of -1 to +1 (inclusive).

Compatibility This function *is* in the C99 specification.

See Also [cosf\(\)](#), [sinf\(\)](#)

tanh Function

Purpose Computes the hyperbolic tangent of *x*.

Declared In `posix/math.h`

Prototype `double tanh (double x)`

Parameters $\rightarrow x$
Value of type `double` to be evaluated. The value of *x* has no range limit.

Returns Returns the hyperbolic tangent of *x*, a value without a range limit.

Compatibility This function *is* in the C99 specification.

See Also [cosh\(\)](#), [sinh\(\)](#)

tanhf Function

Purpose	Computes the hyperbolic tangent of <i>x</i> .
Declared In	posix/math.h
Prototype	float tanhf (float <i>x</i>)
Parameters	→ <i>x</i> Value of type float to be evaluated.
Returns	Returns the hyperbolic tangent of <i>x</i> , a value without a range limit.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	coshf() , sinhf()

tanh1 Function

Purpose	Computes the hyperbolic tangent of <i>x</i> .
Declared In	posix/math.h
Prototype	long double tanh1 (long double <i>x</i>)
Parameters	→ <i>x</i> Value of type long double to be evaluated.
Returns	Returns the hyperbolic tangent of <i>x</i> , a value without a range limit.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	cosh1() , sinh1()

tan1 Function

Purpose	Computes the tangent of <i>x</i> .
Declared In	posix/math.h
Prototype	long double tan1 (long double <i>x</i>)
Parameters	→ <i>x</i> Value of type long double to be evaluated.
Returns	Returns the tangent of <i>x</i> , a value within the range of -1 to +1 (inclusive).
Compatibility	This function <i>is</i> in the C99 specification.
See Also	cos1() , sin1()

math.h

tanl

netdb.h

The <netdb.h> header defines functions useful for network database operations.

Structures and Types

addrinfo Struct

Purpose	This structure contains the information obtained from the address.
Declared In	posix/netdb.h
Prototype	<pre>struct addrinfo { int ai_flags; int ai_family; int ai_socktype; int ai_protocol; size_t ai_addrlen; char *ai_canonname; struct sockaddr *ai_addr; struct addrinfo *ai_next; }</pre>
Fields	<p>ai_flags AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST.</p> <p>ai_family PF_XXX.</p> <p>ai_socktype SOCK_XXX.</p> <p>ai_protocol 0 or IPPROTO_XXX for IPv4 and IPv6.</p> <p>ai_addrlen The length of ai_addr.</p>

netdb.h

hostent

`ai_canonname`
Canonical name for hostname.

`ai_addr`
Binary address.

`ai_next`
Next structure in linked list.

Comments All addresses are supplied in host order and returned in network order (suitable for use in system calls).

hostent Struct

Purpose This structure contains either the information obtained from the name server or database entries supplied by the system.

Declared In `posix/netdb.h`

Prototype

```
struct hostent {  
    char *h_name;  
    char **h_aliases;  
    int h_addrtype;  
    int h_length;  
    char **h_addr_list;  
}
```

Fields `h_name`
Official name of the host.

`h_aliases`
A list of alternative names for the host.

`h_addrtype`
Host address type.

`h_length`
The length, in bytes, of the address.

`h_addr_list`
List of addresses from name server.

Comments All addresses are supplied in host order and returned in network order (suitable for use in system calls).

netent Struct

Purpose	This structure contains the information obtained from the network.
Declared In	posix/netdb.h
Prototype	<pre>struct netent { char *n_name; char **n_aliases; int n_addrtype; unsigned long n_net; }</pre>
Fields	<p>n_name Official name of the network.</p> <p>n_aliases A list of alternative names for the network.</p> <p>n_addrtype Network address type.</p> <p>n_net The network number.</p>
Comments	All addresses are supplied in host order and returned in network order (suitable for use in system calls).

protoent Struct

Purpose	This structure contains the information obtained from the protocol.
Declared In	posix/netdb.h
Prototype	<pre>struct protoent { char *p_name; char **p_aliases; int p_proto; }</pre>
Fields	<p>p_name Official name of the protocol.</p> <p>p_aliases A list of alternative names for the protocol.</p> <p>p_proto The protocol number.</p>

netdb.h

servent

Comments All addresses are supplied in host order and returned in network order (suitable for use in system calls).

servent Struct

Purpose This structure contains the information obtained from the service.

Declared In posix/netdb.h

Prototype

```
struct servent {
    char *s_name;
    char **s_aliases;
    int s_port;
    char *s_proto;
}
```

Fields

s_name
Official name of the service.

s_aliases
A list of alternative names for the service.

s_port
The port number.

s_proto
The protocol to use.

Comments All addresses are supplied in host order and returned in network order (suitable for use in system calls).

Functions and Macros

endhostent Function

Purpose Closes the TCP connection.

Declared In posix/netdb.h

Prototype void endhostent (void)

Compatibility This function is *not* in the C99 specification.

endnetent Function

- Purpose** Closes the connection to the database, releasing any open file descriptor.
- Declared In** `posix/netdb.h`
- Prototype** `void endnetent (void)`
- Compatibility** This function is *not* in the C99 specification.

endprotoent Function

- Purpose** Closes the connection to the database, releasing any open file descriptor.
- Declared In** `posix/netdb.h`
- Prototype** `void endprotoent (void)`
- Compatibility** This function is *not* in the C99 specification.

endservent Function

- Purpose** Closes the connection to the database, releasing any open file descriptor.
- Declared In** `posix/netdb.h`
- Prototype** `void endservent (void)`
- Compatibility** This function is *not* in the C99 specification.

freeaddrinfo Function

- Purpose** Returns the socket address structures and canonical node name strings pointed to by the `addrinfo` structures.
- Declared In** `posix/netdb.h`
- Prototype** `void freeaddrinfo (struct addrinfo *ai)`
- Parameters** $\rightarrow ai$
The `addrinfo` structure pointed to by the `ai` argument is freed, along with any dynamic storage pointed to by the

structure. This operation is repeated until a NULL `ai_next` pointer is encountered.

Compatibility This function is *not* in the C99 specification.

freehostent Function

Purpose Releases the dynamically allocated memory of the `hostent` structure.

Returns Returns a pointer to an object of the `hostent` structure.

Declared In `posix/netdb.h`

Prototype `void freehostent (struct hostent *ip)`

Parameters $\rightarrow ip$
A pointer to an object of the `hostent` structure.

Compatibility This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

gai_strerror Function

Purpose Aids applications in printing error messages based on the `EAI_XXX` codes.

Declared In `posix/netdb.h`

Prototype `const char *gai_strerror (int ecode)`

Parameters $\rightarrow ecode$
An `EAI_XXX` code, such as `EAI_ADDRFAMILY`.

Returns Returns a pointer to a string whose contents indicate an unknown error.

Compatibility This function is *not* in the C99 specification.

getaddrinfo Function

Purpose	Protocol-independent nodename-to-address translation.
Declared In	posix/netdb.h
Prototype	<pre>int getaddrinfo (const char *nodename, const char *servname, const struct addrinfo *hints, struct addrinfo **res)</pre>
Parameters	<p>→ <i>nodename</i> A pointer to null-terminated strings or NULL.</p> <p>→ <i>servname</i> A pointer to null-terminated strings or NULL.</p> <p>→ <i>hints</i> Hints concerning the type of socket that the caller supports.</p> <p>← <i>res</i> A pointer to a linked list of one or more <code>addrinfo</code> structures.</p>
Returns	Returns a set of socket addresses and associated information to be used in creating a socket with which to address the specified service.
Comments	<p>One or both of the <i>nodename</i> and <i>servname</i> parameters must be a non-NULL pointer.</p> <p>If <i>nodename</i> is not NULL, the requested service location is named by <i>nodename</i>; otherwise, the requested service location is local to the caller. If <i>servname</i> is NULL, the call returns network-level addresses for the specified <i>nodename</i>. If <i>servname</i> is not NULL, it is a null-terminated character string identifying the requested service.</p>
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	gethostbyname() , getservbyname()

gethostbyaddr Function

- Purpose** Searches for the specified host in the current domain and its parents unless the name ends in a dot.
- Declared In** `posix/netdb.h`
- Prototype** `struct hostent *gethostbyaddr (const char *addr, int len, int type)`
- Parameters**
- *addr*
Host address type.
 - *len*
The length, in bytes, of the address.
 - *type*
A named constant that indicates the naming scheme under which the lookup is performed. Must be specified as `AF_INET`.
- Returns** Returns a pointer to an object of the `hostent` structure, describing an Internet host referenced by address.
- Compatibility** This function is *not* in the C99 specification.

gethostbyname Function

- Purpose** Searches for the specified host in the current domain and its parents unless the name ends in a dot.
- Declared In** `posix/netdb.h`
- Prototype** `struct hostent *gethostbyname (const char *name)`
- Parameters**
- *name*
Official name of the host.
- Returns** Returns a pointer to an object of the `hostent` structure, describing an Internet host referenced by name.
- Compatibility** This function is *not* in the C99 specification.

gethostbyname2 Function

Purpose	An evolution of <code>gethostbyname()</code> that allows lookups in address families other than <code>AF_INET</code> .
Declared In	<code>posix/netdb.h</code>
Prototype	<pre>struct hostent *gethostbyname2 (const char *name, int af)</pre>
Parameters	<p>→ <i>name</i> Official name of the host.</p> <p>→ <i>af</i> Must be specified as <code>AF_INET</code> or <code>AF_INET6</code>.</p>
Returns	Returns a pointer to an object of the <code>hostent</code> structure, describing an Internet host referenced by name.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).

gethostent Function

Purpose	Reads the next entry in the database, opening and closing a connection to the database as necessary.
Declared In	<code>posix/netdb.h</code>
Prototype	<pre>struct hostent *gethostent (void)</pre>
Returns	Returns a pointer to an object of the <code>hostent</code> structure.
Compatibility	This function is <i>not</i> in the C99 specification.

getipnodebyaddr Function

Purpose	Returns the address of a network host.
Declared In	<code>posix/netdb.h</code>
Prototype	<pre>struct hostent *getipnodebyaddr (const void *src, size_t len, int af, int *error_num)</pre>
Parameters	<p>→ <i>src</i> The name of the host whose network address to look up.</p>

netdb.h

getipnodebyname

→ *len*

The length, in bytes, of the address.

→ *af*

Must be specified as AF_INET or AF_INET6.

← *error_num*

A NULL pointer is returned if an error occurred, and *error_num* contains an error code from the following list: HOST_NOT_FOUND, NO_ADDRESS, NO_RECOVERY, or TRY_AGAIN.

Returns Returns a pointer to an object of the `hostent` structure, describing an Internet host referenced by address.

Compatibility This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

getipnodebyname Function

Purpose Returns the name of a network host.

Declared In `posix/netdb.h`

Prototype

```
struct hostent *getipnodebyname
    (const char *name, int af, int flags,
     int *error_num)
```

Parameters → *name*

Official name of the host.

→ *af*

Must be specified as AF_INET or AF_INET6.

→ *flags*

Specifies additional options: AI_V4MAPPED, AI_ALL, or AI_ADDRCONFIG. More than one option can be specified by logically ORing them together. *flags* should be set to zero (0) if no options are desired.

← *error_num*

A NULL pointer is returned if an error occurred, and *error_num* contains an error code from the following list: HOST_NOT_FOUND, NO_ADDRESS, NO_RECOVERY, or TRY_AGAIN.

- Returns** Returns a pointer to an object of the `hostent` structure, describing an Internet host referenced by name.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

getnameinfo Function

- Purpose** Translates address-to-nodename in a protocol-independent manner.
- Declared In** `posix/netdb.h`
- Prototype**

```
int getnameinfo (const struct sockaddr *sa,  
                size_t salen, char *host, size_t hostlen,  
                char *serv, size_t servlen, int flags)
```
- Parameters**
- *sa*
A `sockaddr` structure.
 - *salen*
The length, in bytes, of the `sockaddr` structure.
 - *host*
The buffer that holds the IP address.
 - *hostlen*
The length, in bytes, of the IP address buffer.
 - *serv*
The buffer that holds the port number.
 - *servlen*
The length, in bytes, of the port number buffer.
 - *flags*
Changes the default actions of this function.
- Returns** Returns text strings for the IP address and port number in user-provided buffers.
- Compatibility** This function is *not* in the C99 specification.

getnetbyaddr Function

- Purpose** Searches from the beginning of the file until a matching network address is found, or until EOF is encountered.
- Declared In** `posix/netdb.h`
- Prototype** `struct netent *getnetbyaddr (unsigned long net,
int type)`
- Parameters** `→ net`
The network number.
`→ type`
Network address type.
- Returns** Returns a pointer to an object of the `netent` structure, describing the network database.
- Compatibility** This function is *not* in the C99 specification.

getnetbyname Function

- Purpose** Searches from the beginning of the file until a matching network name is found, or until EOF is encountered.
- Declared In** `posix/netdb.h`
- Prototype** `struct netent *getnetbyname (const char *name)`
- Parameters** `→ name`
Official name of the network.
- Returns** Returns a pointer to an object of the `netent` structure, describing the network database.
- Compatibility** This function is *not* in the C99 specification.

getnetent Function

- Purpose** Reads the next line of the file, opening the file if necessary.
- Declared In** `posix/netdb.h`
- Prototype** `struct netent *getnetent (void)`
- Returns** Returns a pointer to an object of the `netent` structure, describing the network database.
- Compatibility** This function is *not* in the C99 specification.

getprotobyname Function

- Purpose** Sequentially searches from the beginning of the file until a matching protocol name is found, or until EOF is encountered.
- Declared In** `posix/netdb.h`
- Prototype** `struct protoent *getprotobyname
(const char *name)`
- Parameters** → *name*
Official name of the protocol.
- Returns** Returns a pointer to an object of the `protoent` structure, describing the network database.
- Compatibility** This function is *not* in the C99 specification.

getprotobynumber Function

- Purpose** Sequentially searches from the beginning of the file until a matching protocol number is found, or until EOF is encountered.
- Declared In** `posix/netdb.h`
- Prototype** `struct protoent *getprotobynumber (int proto)`
- Parameters** → *proto*
Official name of the protocol.
- Returns** Returns a pointer to an object of the `protoent` structure, describing the network database.
- Compatibility** This function is *not* in the C99 specification.

getprotoent Function

- Purpose** Reads the next line of the file, opening the file if necessary.
- Declared In** `posix/netdb.h`
- Prototype** `struct protoent *getprotoent (void)`
- Returns** Returns a pointer to an object of the `protoent` structure, describing the network database.
- Compatibility** This function is *not* in the C99 specification.

getservbyname Function

- Purpose** Searches from the beginning of the file until a matching protocol name is found, or until EOF is encountered.
- Declared In** `posix/netdb.h`
- Prototype** `struct servent *getservbyname (const char *name, const char *proto)`
- Parameters**
- *name*
Official name of the network.
 - *proto*
The protocol.
- Returns** Returns a pointer to an object of the `servent` structure, describing the network services database.
- Compatibility** This function is *not* in the C99 specification.

getservbyport Function

- Purpose** Searches from the beginning of the file until a matching port number is found, or until EOF is encountered.
- Declared In** `posix/netdb.h`
- Prototype** `struct servent *getservbyport (int port, const char *proto)`
- Parameters**
- *port*
The port number.
 - *proto*
The protocol to use
- Returns** Returns a pointer to an object of the `servent` structure, describing the network services database.
- Compatibility** This function is *not* in the C99 specification.

getservent Function

- Purpose** Reads the next line of the file, opening the file if necessary.
- Declared In** `posix/netdb.h`
- Prototype** `struct servent *getservent (void)`
- Returns** Returns a pointer to an object of the `servent` structure, describing the network services database.
- Compatibility** This function is *not* in the C99 specification.

hstrerror Function

- Purpose** Returns a string that is the message text corresponding to the value of the `err` parameter.
- Declared In** `posix/netdb.h`
- Prototype** `const char *hstrerror (int err)`
- Parameters** → `err`
The error.
- Returns** Returns a string that is the message text corresponding to the value of the `err` parameter.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

sethostent Function

- Purpose** Requests the use of a connected TCP socket for queries.
- Declared In** `posix/netdb.h`
- Prototype** `void sethostent (int stayopen)`
- Parameters** → `stayopen`
If the `stayopen` flag is non-zero, sets the option to send all queries to the name server using TCP and to retain the connection after each call to `gethostbyname ()`, `gethostbyname2 ()`, or `gethostbyaddr ()`. Otherwise, queries are performed using UDP datagrams.

netdb.h

setnetent

Compatibility This function is *not* in the C99 specification.

See Also [gethostbyaddr\(\)](#), [gethostbyname\(\)](#), [gethostbyname2\(\)](#)

setnetent Function

Purpose Opens and rewinds a file.

Declared In `posix/netdb.h`

Prototype `void setnetent (int stayopen)`

Parameters `→ stayopen`

If non-zero, the network database is not closed after each call to `getnetbyname()` or `getnetbyaddr()`.

Compatibility This function is *not* in the C99 specification.

See Also [getnetbyaddr\(\)](#), [getnetbyname\(\)](#)

setprotoent Function

Purpose Opens and rewinds a file.

Declared In `posix/netdb.h`

Prototype `void setprotoent (int stayopen)`

Parameters `→ stayopen`

If non-zero, the network database is not closed after each call to `getprotobyname()` or `getprotobynumber()`.

Compatibility This function is *not* in the C99 specification.

See Also [getprotobyname\(\)](#), [getprotobynumber\(\)](#)

setservent Function

Purpose	Opens and rewinds a file.
Declared In	posix/netdb.h
Prototype	void setservent (int <i>stayopen</i>)
Parameters	→ <i>stayopen</i> If non-zero, the network database is not closed after each call to <code>getservbyname()</code> or <code>getservbyport()</code> .
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	getservbyname() , getservbyport()

netdb.h

setservernt

PalmMath.h

The <PalmMath.h> header defines Palm OS specific mathematical functions not specified in the ANSI/ISO standard.

Constants

Math Constants

Purpose These constants are intended to be used as 32-bit floats. These constants should not be used as double precision arguments. However, a new double precision version of each of these may be created by removing the “f” suffix from the end of each decimal string.

Declared In posix/sys/palmmath.h

Constants

```
#define M_E 2.7182818284590452354f
    Approximates the mathematical constant e.
#define M_LOG2E 1.4426950408889634074f
    Approximates the mathematical constant log2(e).
#define M_LOG10E 0.43429448190325182765f
    Approximates the mathematical constant log10(e).
#define M_LN2 0.69314718055994530942f
    Approximates the mathematical constant loge(2).
#define M_LN10 2.30258509299404568402f
    Approximates the mathematical constant loge(10).
#define M_PI 3.14159265358979323846f
    Single precision approximation to π.
#define M_PI_2 1.57079632679489661923f
    Single precision approximation to π/2.
#define M_1_PI 0.31830988618379067154f
    Single precision approximation to 1/π.
```

PalmMath.h

Math Constants

```
#define M_PI_4 0.78539816339744830962f
    Single precision approximation to  $\pi/4$ .
#define M_2_PI 0.63661977236758134308f
    Single precision approximation to  $2/\pi$ .
#define M_2_SQRTPI 1.12837916709551257390f
    Single precision approximation to  $2/\sqrt{\pi}$ .
#define M_SQRT2 1.41421356237309504880f
    Approximates the mathematical constant  $\sqrt{2}$ .
#define M_SQRT1_2 0.70710678118654752440f
    Approximates the mathematical constant  $1/\sqrt{2}$ .
#define PI M_PI
    Single precision approximation to  $\pi$ .
#define PI2 M_PI_2
    Single precision approximation to  $\pi/2$ .
#define M_PI_3 1.047197551196597746154f
    Single precision approximation to  $\pi/3$ .
#define M_3_PI_4 2.356194490192344928846f
    Single precision approximation to  $3*\pi/4$ .
#define M_5_PI_4 3.926990816987241548076f
    Single precision approximation to  $5*\pi/4$ .
#define M_3_PI_2 4.71238898038468985769f
    Single precision approximation to  $3*\pi/2$ .
#define M_7_PI_4 5.497787143782138167306f
    Single precision approximation to  $7*\pi/4$ .
```

Functions and Macros

lceilf Function

Purpose	Computes the nearest 32-bit signed integer not less than x .
Declared In	<code>posix/sys/palmmath.h</code>
Prototype	<code>int32_t lceilf (float x)</code>
Parameters	$\rightarrow x$ Value of type <code>float</code> to be evaluated.
Returns	Returns the nearest 32-bit signed integer not less than x . In cases where x is out of the range of representable integers, $+/-INT_MAX$ is returned.
Comments	Exceptions are never raised.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).
See Also	ceil() , ceilf() , lfloorf()

lfloorf Function

Purpose	Computes the nearest 32-bit signed integer not greater than x .
Declared In	<code>posix/sys/palmmath.h</code>
Prototype	<code>int32_t lfloorf (float x)</code>
Parameters	$\rightarrow x$ Value of type <code>float</code> to be evaluated.
Returns	Returns the nearest 32-bit signed integer not greater than x . In cases where x is out of the range of representable integers, $+/-INT_MAX$ is returned.
Comments	Exceptions are never raised.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).
See Also	lceilf() , floor() , floorf()

sincosf Function

- Purpose** Computes an approximation to the sine (*sin_val*) and cosine (*cos_val*) of any angle in a single call.
- Declared In** `posix/sys/palmmath.h`
- Prototype** `void sincosf (float angle, float *cos_val, float *sin_val)`
- Parameters**
- *angle*
Must be specified in radians.
 - *cos_val*
Cosine value.
 - *sin_val*
Sine value.
- Returns** Returns the approximation to the sine (*sin_val*) and cosine (*cos_val*) of the specified angle.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).
- See Also** [cos\(\)](#), [sin\(\)](#)

select.h

The `<select.h>` header defines the `select()` macro, which is used for synchronous I/O multiplexing.

Functions and Macros

select Function

Purpose	Examines the I/O descriptor sets whose addresses are passed in to see if some of their descriptors are ready.
Declared In	<code>posix/sys/select.h</code>
Prototype	<pre>int select (int <i>fd</i>, fd_set *<i>rfd</i>s, fd_set *<i>wfd</i>s, fd_set *<i>efd</i>s, struct timeval *<i>timeout</i>)</pre>
Parameters	<p>→ <i>fd</i></p> <p>The descriptors are checked in each set; that is, the descriptors from zero (0) through <i>fd</i> - 1 in the descriptor sets are examined.</p> <p>→ <i>rfd</i>s</p> <p>The descriptors are checked to see if some of them are ready for reading.</p> <p>→ <i>wfd</i>s</p> <p>The descriptors are checked to see if some of them are ready for writing.</p> <p>→ <i>efd</i>s</p> <p>The descriptors are checked to see if some of them have an exceptional condition pending.</p> <p>→ <i>timeout</i></p> <p>If <i>timeout</i> is a non-NULL pointer, it specifies a maximum interval to wait for the selection to complete. If <i>timeout</i> is a NULL pointer, then <code>select()</code> blocks indefinitely. To affect a</p>

select.h

select

poll, the *timeout* argument should be non-NULL, pointing to a zero-valued `timeval` structure.

Returns Returns the number of ready descriptors that are contained in the descriptor sets. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error. If the time limit expires, `select()` returns zero (0). If `select()` returns with an error, including one due to an interrupted call, the descriptor sets are unmodified.

Compatibility This function is *not* in the C99 specification.

See Also [accept\(\)](#), [connect\(\)](#), [read\(\)](#), [recv\(\)](#), [send\(\)](#), [write\(\)](#)

socket.h

The `<socket.h>` header defines several functions useful to sockets.

Structures and Types

sockaddr Struct

Purpose	Defines a structure used by the kernel to store most addresses.				
Declared In	<code>posix/sys/time.h</code>				
Prototype	<pre>struct sockaddr { sa_family_t sa_family; char sa_data[14]; }</pre>				
Fields	<table><tr><td><code>sa_family</code></td><td>The address family.</td></tr><tr><td><code>sa_data</code></td><td>The address value.</td></tr></table>	<code>sa_family</code>	The address family.	<code>sa_data</code>	The address value.
<code>sa_family</code>	The address family.				
<code>sa_data</code>	The address value.				

socklen_t Typedef

Purpose	Definitions related to sockets: types, address families, options.
Declared In	<code>posix/sys/socket.h</code>
Prototype	<code>typedef unsigned int socklen_t</code>

Functions and Macros

accept Function

Purpose	Accepts a connection on a socket by extracting the first connection request on the queue of pending connections, creating a new socket with the same properties of <i>sock</i> and allocating a new file descriptor for the socket.
Declared In	<code>posix/sys/socket.h</code>
Prototype	<code>int accept (int <i>sock</i>, struct sockaddr *<i>addr</i>, socklen_t *<i>addrlen</i>)</code>
Parameters	<p>→ <i>sock</i> A socket that has been created with <code>socket()</code>, bound to an address with <code>bind()</code>, and listening for connections after a <code>listen()</code>.</p> <p>← <i>addr</i> A result parameter that is filled in with the source address of the connecting entity, as known to the communications layer.</p> <p>↔ <i>addrlen</i> Initially contains the amount of space pointed to by <i>addr</i>; on return, it contains the actual length (in bytes) of the address returned.</p>
Returns	Returns a non-negative integer that is a descriptor for the accepted socket. Otherwise, -1 is returned and the global variable <code>errno</code> is set to indicate the error.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	bind() , connect() , listen() , select() , socket()

bind Function

Purpose	Assigns a name to an unnamed socket.
Declared In	posix/sys/socket.h
Prototype	<pre>int bind (int sock, const struct sockaddr *addr, socklen_t addrlen)</pre>
Parameters	<p>→ <i>sock</i> A socket that has been created with <code>socket ()</code> that exists in a namespace but has no name defined.</p> <p>← <i>addr</i> A result parameter that is filled in with the source address of the connecting entity, as known to the communications layer.</p> <p>↔ <i>addrlen</i> Initially contains the amount of space pointed to by <i>addr</i>; on return, it contains the actual length (in bytes) of the address returned.</p>
Returns	Returns zero (0) if the bind is successful. Otherwise, -1 is returned and the global variable <code>errno</code> is set to indicate the error.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	connect () , getsockname () , listen () , socket ()

connect Function

Purpose	Initiates a connection on a socket.
Declared In	posix/sys/socket.h
Prototype	<pre>int connect (int sock, const struct sockaddr *addr, socklen_t addrlen)</pre>
Parameters	<p>→ <i>sock</i> A socket.</p> <p>← <i>addr</i> A result parameter that is filled in with the source address of the connecting entity, as known to the communications layer.</p>

socket.h

getpeername

↔ *addrlen*

Initially contains the amount of space pointed to by *addr*; on return, it contains the actual length (in bytes) of the address returned.

Returns Returns zero (0) if the connection or binding is successful. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

See Also [accept\(\)](#), [getsockname\(\)](#), [getsockopt\(\)](#), [select\(\)](#), [socket\(\)](#)

getpeername Function

Purpose Gets the name of the connected peer.

Declared In `posix/sys/socket.h`

Prototype `int getpeername (int sock, struct sockaddr *addr, socklen_t addrlen)`

Parameters → *sock*
A socket.

← *addr*
A result parameter that is filled in with the source address of the connecting entity, as known to the communications layer.

↔ *addrlen*
Initially contains the amount of space pointed to by *addr*; on return, it contains the actual length (in bytes) of the address returned.

Returns Returns the name of the peer connected to the specified socket.

Compatibility This function is *not* in the C99 specification.

See Also [accept\(\)](#), [bind\(\)](#), [getsockname\(\)](#), [socket\(\)](#)

getsockname Function

Purpose	Gets the socket name.
Declared In	posix/sys/socket.h
Prototype	int getsockname (int <i>sock</i> , struct sockaddr * <i>addr</i> , socklen_t <i>addrlen</i>)
Parameters	→ <i>sock</i> A socket. ← <i>addr</i> A result parameter that is filled in with the source address of the connecting entity, as known to the communications layer. ↔ <i>addrlen</i> Initially contains the amount of space pointed to by <i>addr</i> ; on return, it contains the actual length (in bytes) of the address returned.
Returns	Returns the current name for the specified socket.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	bind() , socket()

getsockopt Function

Purpose	Gets the options on sockets.
Declared In	posix/sys/socket.h
Prototype	int getsockopt (int <i>sock</i> , int <i>level</i> , int <i>option</i> , void * <i>optval</i> , socklen_t * <i>optlen</i>)
Parameters	→ <i>sock</i> A socket. → <i>level</i> To manipulate options at the socket level, <i>level</i> is specified as SOL_SOCKET. → <i>option</i> <i>option</i> and any specified options are passed uninterpreted to the appropriate protocol module for interpretation.

socket.h

listen

Returns Returns zero (0) if the connection or binding is successful. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

See Also [getprotoent\(\)](#), [ioctl\(\)](#), [select\(\)](#), [socket\(\)](#), [setsockopt\(\)](#)

listen Function

Purpose Listens for connections on a socket.

Declared In `posix/sys/socket.h`

Prototype `int listen (int sock, int backlog)`

Parameters → *sock*

A socket.

→ *backlog*

The maximum length the queue of pending connections may grow to.

Returns Returns zero (0) if the connection or binding is successful. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

See Also [accept\(\)](#), [connect\(\)](#), [socket\(\)](#)

recv Function

Purpose Normally used only on a connected socket and is identical to `recvfrom()` with a NULL *addr* parameter.

Declared In `posix/sys/socket.h`

Prototype `ssize_t recv (int sock, void *data,
size_t datalen, int flags)`

Parameters → *sock*

A socket.

→ *data*

The message.

→ *datalen*

The length of the message.

→ *flags*

ORs together one or more of the values: MSG_OOB, MSG_PEEK, MSG_WAITALL.

Returns Returns the length of the message upon successful completion. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from.

Compatibility This function is *not* in the C99 specification.

See Also [connect\(\)](#), [recvfrom\(\)](#), [recvmsg\(\)](#)

recvfrom Function

Purpose Receives messages from a socket, and may be used to receive data on a socket whether or not it is connection-oriented.

Declared In `posix/sys/socket.h`

Prototype

```
ssize_t recvfrom (int sock, void *data,
                 size_t datalen, int flags,
                 struct sockaddr *addr, socklen_t *addrlen)
```

Parameters → *sock*
A socket.

→ *data*
The message.

→ *datalen*
The length of the message.

→ *flags*
ORs together one or more of the values: MSG_OOB, MSG_PEEK, MSG_WAITALL.

→ *addr*
If *addr* is non-NULL, and the socket is not connection-oriented, the source address of the message is filled in.

socket.h

recvmsg

← *addrlen*

Initially contains the amount of space pointed to by *addr*; on return, it contains the actual length (in bytes) of the address stored there.

Returns Returns the length of the message upon successful completion. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from.

Compatibility This function is *not* in the C99 specification.

See Also [connect\(\)](#), [recv\(\)](#), [recvmsg\(\)](#)

recvmsg Function

Purpose Receives messages from a socket, and may be used to receive data on a socket whether or not it is connection-oriented.

Declared In `posix/sys/socket.h`

Prototype `ssize_t recvmsg (int sd, struct msghdr *msg, int flags)`

Parameters → *sd*
A socket.

→ *msg*
The message.

→ *flags*
ORs together one or more of the values: `MSG_OOB`, `MSG_PEEK`, `MSG_WAITALL`.

Returns Returns the length of the message upon successful completion. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from.

Compatibility This function is *not* in the C99 specification.

See Also [connect\(\)](#), [recv\(\)](#), [recvfrom\(\)](#)

send Function

Purpose	Sends a message from a socket.
Declared In	posix/sys/socket.h
Prototype	<pre>ssize_t send (int sock, const void *data, size_t datalen, int flags)</pre>
Parameters	<p>→ <i>sock</i> A socket.</p> <p>→ <i>data</i> The message.</p> <p>→ <i>datalen</i> The length of the message.</p> <p>→ <i>flags</i> ORs together one or more of the values: MSG_OOB, MSG_DONTROUTE.</p>
Returns	Returns the number of characters sent. Otherwise, -1 is returned and the global variable <code>errno</code> is set to indicate the error.
Comments	May be used only when the socket is in a connected state.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	select() , sendmsg() , sendto()

sendmsg Function

Purpose	Sends a message from a socket.
Declared In	posix/sys/socket.h
Prototype	<pre>ssize_t sendmsg (int sd, const struct msghdr *msg, int flags)</pre>
Parameters	<p>→ <i>sd</i> A socket.</p> <p>→ <i>msg</i> The message.</p> <p>→ <i>flags</i> ORs together one or more of the values: MSG_OOB, MSG_DONTROUTE.</p>

socket.h

sendto

- Returns** Returns the number of characters sent. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [select\(\)](#), [send\(\)](#), [sendto\(\)](#)

sendto Function

- Purpose** Sends a message from a socket.
- Declared In** `posix/sys/socket.h`
- Prototype**

```
ssize_t sendto (int sock, const void *data,  
               size_t datalen, int flags,  
               const struct sockaddr *addr,  
               socklen_t addrlen)
```
- Parameters**
- *sock*
A socket.
 - *data*
The message.
 - *datalen*
The length of the message.
 - *flags*
ORs together one or more of the values: `MSG_OOB`, `MSG_DONTROUTE`.
 - *addr*
If *addr* is non-NULL, and the socket is not connection-oriented, the source address of the message is filled in.
 - ← *addrlen*
Initially contains the amount of space pointed to by *addr*; on return, it contains the actual length (in bytes) of the address stored there.
- Returns** Returns the number of characters sent. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [select\(\)](#), [send\(\)](#), [sendmsg\(\)](#)

setsockopt Function

Purpose	Sets options on sockets.
Declared In	posix/sys/socket.h
Prototype	<pre>int setsockopt (int sock, int level, int option, const void *optval, socklen_t optlen)</pre>
Parameters	<p>→ <i>sock</i> A socket.</p> <p>→ <i>level</i> To manipulate options at the socket level, <i>level</i> is specified as SOL_SOCKET.</p> <p>→ <i>option</i> Any specified option(s) passed uninterpreted to the appropriate protocol module for interpretation.</p> <p>→ <i>optval</i> Used to access option values. Identifies a buffer in which the value for the requested option is returned.</p> <p>→ <i>optlen</i> Used to access option values. Identifies a buffer in which the length for the requested option is returned.</p>
Returns	Returns zero (0) if the connection or binding is successful. Otherwise, -1 is returned and the global variable <code>errno</code> is set to indicate the error.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	getprotoent() , getsockopt() , ioctl() , select() , socket()

shutdown Function

Purpose	Disables subsequent send and/or receive operations on a socket.
Declared In	posix/sys/socket.h
Prototype	<pre>int shutdown (int sock, int direction)</pre>
Parameters	<p>→ <i>sock</i> A socket.</p>

socket.h

socket

→ *direction*

Specifies the type of shutdown. The values are as follows:

SHUT_RD

Disables further receive operations.

SHUT_WR

Disables further send operations.

SHUT_RDWR

Disables further send and receive operations.

Returns Returns zero (0) upon successful completion. Otherwise, 1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

socket Function

Purpose Creates an endpoint for communication.

Declared In `posix/sys/socket.h`

Prototype `int socket (int family, int type, int proto)`

Parameters → *family*

A communications domain within which communication takes place; this selects the protocol family that should be used.

→ *type*

The semantics of communication.

→ *proto*

A particular protocol to be used with the socket.

Returns Returns a descriptor referencing the socket. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

See Also [getsockopt\(\)](#)

stdarg.h

The `<stdarg.h>` header defines several macros useful in the creation of functions that accept a variable number of arguments.

Functions and Macros

va_arg Macro

Purpose	Expands to an expression that has the type and value of the next argument in the call.
Declared In	<code>posix/stdarg.h</code>
Prototype	<code>#define va_arg (va_list ap, t)</code>
Parameters	<code>→ ap</code> An object of type <code>va_list</code> initialized by <code>va_start()</code> . <code>→ t</code> A type.
Returns	Returns an argument value.
See Also	<code>va_start()</code>

va_copy Macro

Purpose	Makes <code>dest</code> a copy of <code>src</code> .
Declared In	<code>posix/stdarg.h</code>
Prototype	<code>#define va_copy (va_list dest, va_list src)</code>
Parameters	<code>→ dest</code> A copy of <code>src</code> . <code>→ src</code> An object of type <code>va_list</code> initialized by <code>va_start()</code> .
Returns	Returns no value.

va_end Macro

- Purpose** Handles a normal function return from the function whose variable argument list was initialized by `va_start()` or `va_copy()`.
- Declared In** `posix/stdarg.h`
- Prototype** `#define va_end (va_list ap)`
- Parameters** `→ ap`
An object of type `va_list` initialized by `va_start()`.
- Returns** Returns no value.
- See Also** [va_copy\(\)](#), [va_start\(\)](#)

va_start Macro

- Purpose** Initializes the variable-length argument list.
- Declared In** `posix/stdarg.h`
- Prototype** `#define va_start (va_list ap, v)`
- Parameters** `→ ap`
An object of type `va_list`.
- `→ v`
The last known fixed argument being passed to the function (the argument before the ellipsis).
- Returns** Returns no value.
- Comments** This macro initializes `ap` for subsequent use by `va_arg()`, `va_copy()`, and `va_end()`, and must be called first.
- See Also** [va_arg\(\)](#), [va_end\(\)](#)

stddef.h

The `<stddef.h>` header defines the commonly used `offsetof()` macro.

Functions and Macros

offsetof Macro

Purpose	Expands to an integer constant expression that has type <code>size_t</code> , the value of which is the offset in bytes to the structure member (designated by <i>member</i>) from the beginning of its structure (designated by <i>type</i>).
Declared In	<code>posix/stddef.h</code>
Prototype	<code>#define offsetof (type, member)</code>
Parameters	<code>→ type</code> The structure. <code>→ member</code> The structure member.
Returns	Returns the offset of a structure's member.

stddef.h

offsetof

stdio.h

The `<stdio.h>` header defines functions for performing input and output.

The current expected behavior of the standard I/O library is to direct `stdout` and `stderr` output to a debugger via `DbgMessage()`, and to read bytes from `stdin` via the debugger using `DbgGetChar()`. Attempting to close one of the standard files [`stdin/stdout/stderr`] is not currently supported.

Functions and Macros

asprintf Function

Purpose	Writes to a dynamically allocated string that is stored in <i>ret</i> .
Declared In	<code>posix/stdio.h</code>
Prototype	<code>int asprintf (char **ret, const char *format, ...)</code>
Parameters	<code>→ ret</code> A dynamically allocated string. <code>→ format</code> A string that specifies how subsequent arguments are converted for output.
Returns	Returns a pointer to a buffer sufficiently large to hold the string in the <i>ret</i> argument.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).

stdio.h

clearerr

This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.

See Also [fprintf\(\)](#), [printf\(\)](#), [snprintf\(\)](#), [sprintf\(\)](#), [vasprintf\(\)](#), [vfprintf\(\)](#), [vprintf\(\)](#), [vsnprintf\(\)](#), [vsprintf\(\)](#)

clearerr Function

Purpose Clears a stream's end-of-file and error status for a stream.

Declared In posix/stdio.h

Prototype void clearerr (FILE **stream*)

Parameters → *stream*
The specified stream.

Compatibility This function *is* in the C99 specification.

See Also [feof\(\)](#), [ferror\(\)](#), [fileno\(\)](#)

fclose Function

Purpose Closes a stream.

Declared In posix/stdio.h

Prototype int fclose (FILE **stream*)

Parameters → *stream*
The specified stream.

Returns Returns zero (0) upon successful completion.

Comments This function disassociates the specified stream from its underlying file or set of functions. If the stream was being used for output, any buffered data is written first.

Compatibility This function *is* in the C99 specification.

See Also [fflush\(\)](#)

fdopen Function

Purpose	Associates a stream with the existing file descriptor.
Declared In	posix/stdio.h
Prototype	FILE *fdopen (int <i>fileds</i> , const char * <i>mode</i>)
Parameters	→ <i>fileds</i> The existing file descriptor. → <i>mode</i> Must be compatible with the mode of the file descriptor.
Returns	Returns a FILE pointer upon successful completion. Otherwise, NULL is returned and the global variable errno is set to indicate the error.
Comments	The stream is positioned at the file offset of the file descriptor.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	fopen() , freopen()

feof Function

Purpose	Checks the end-of-file status of a stream.
Declared In	posix/stdio.h
Prototype	int feof (FILE * <i>stream</i>)
Parameters	→ <i>stream</i> The specified stream.
Returns	Returns non-zero if the end-of-file indicator is set.
Comments	The end-of-file indicator can only be cleared by the function clearerr().
Compatibility	This function <i>is</i> in the C99 specification.
See Also	clearerr() , ferror() , fileno()

ferror Function

- Purpose** Checks the error status of a stream.
- Declared In** `posix/stdio.h`
- Prototype** `int ferror (FILE *stream)`
- Parameters** `→ stream`
The specified stream.
- Returns** Returns non-zero if the error indicator is set.
- Comments** The end-of-file indicator can only be cleared by the function `clearerr()`.
- Compatibility** This function *is* in the C99 specification.
- See Also** [clearerr\(\)](#), [feof\(\)](#), [fileno\(\)](#)

fflush Function

- Purpose** Flushes a stream.
- Declared In** `posix/stdio.h`
- Prototype** `int fflush (FILE *stream)`
- Parameters** `→ stream`
The specified stream.
- Returns** Returns zero (0) upon successful completion. Otherwise, EOF is returned and the global variable `errno` is set to indicate the error.
- Comments** The open status of the stream is unaffected.
- Compatibility** This function *is* in the C99 specification.
- See Also** [fclose\(\)](#), [fpurge\(\)](#)

fgetc Function

- Purpose** Gets a character from a stream.
- Declared In** `posix/stdio.h`
- Prototype** `int fgetc (FILE *stream)`

- Parameters** → *stream*
The specified stream.
- Returns** Returns the next requested object from the stream. Otherwise, EOF is returned if the stream is at end-of-file or a read error occurs.
- Compatibility** This function *is* in the C99 specification.
- See Also** [getc\(\)](#), [ungetc\(\)](#)

fgetln Function

- Purpose** Gets a line from a stream.
- Declared In** posix/stdio.h
- Prototype** `char *fgetln (FILE *stream, size_t *len)`
- Parameters** → *stream*
The specified stream.
- *len*
The length of the line, including the final newline.
- Returns** Returns a pointer to the line upon successful completion. Otherwise, NULL is returned.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).
- See Also** [fgets\(\)](#)

fgetpos Function

- Purpose** Gets a file position for a stream.
- Declared In** posix/stdio.h
- Prototype** `int fgetpos (FILE *stream, fpos_t *pos)`
- Parameters** → *stream*
The specified stream.
- *pos*
The current value of the file offset from the object.

stdio.h

fgets

- Returns** Returns zero (0) upon successful completion. Otherwise, a non-zero value is returned and the global variable `errno` is set to indicate the error.
- Comments** An alternative interface equivalent to `ftell()` and `ftello()`.
- Compatibility** This function *is* in the C99 specification.
- See Also** [ftell\(\)](#), [ftello\(\)](#)

fgets Function

- Purpose** Gets a line from a stream.
- Declared In** `posix/stdio.h`
- Prototype** `char *fgets (char *str, int size, FILE *stream)`
- Parameters**
- *str*
A character string.
 - *size*
The number of characters to look for.
 - *stream*
The specified stream.
- Returns** Returns a pointer to the string upon successful completion. Otherwise, `NULL` is returned if the stream is at end-of-file or a read error occurs before any characters are read.
- Comments** Reads at most one less than the number of characters specified by *size* from the specified stream and stores the characters in the string *str*. Reading stops when a newline character is found, at end-of-file, or error. This function does not distinguish between end-of-file and error.
- Compatibility** This function *is* in the C99 specification.
- See Also** [fgetln\(\)](#), [fgets\(\)](#)

fileno Function

Purpose	Examines the argument <i>stream</i> and returns its integer descriptor.
Declared In	posix/stdio.h
Prototype	int fileno (FILE * <i>stream</i>)
Parameters	→ <i>stream</i> The specified stream.
Returns	Returns an integer descriptor.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	clearerr() , feof() , ferror()

fopen Function

Purpose	Opens the file whose name is the string pointed to by <i>path</i> and associates a <i>stream</i> with it.
Declared In	posix/stdio.h
Prototype	FILE *fopen (const char * <i>path</i> , const char * <i>mode</i>)
Parameters	→ <i>path</i> A path pointing to a string containing a file name. → <i>mode</i> A string indicating the mode.
Returns	Returns a FILE pointer upon successful completion. Otherwise, NULL is returned and the global variable <code>errno</code> is set to indicate the error.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	fdopen() , freopen()

fprintf Function

- Purpose** Writes formatted output to an output stream.
- Declared In** `posix/stdio.h`
- Prototype** `int fprintf (FILE *stream, const char *format, ...)`
- Parameters**
- *stream*
The specified stream.
 - *format*
A string that specifies how subsequent arguments are converted for output.
- Returns** Returns the number of characters transmitted, or a negative value if an output or encoding error occurred.
- Compatibility** This function *is* in the C99 specification.
This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.
- See Also** [asprintf\(\)](#), [printf\(\)](#), [snprintf\(\)](#), [sprintf\(\)](#), [vasprintf\(\)](#), [vfprintf\(\)](#), [vprintf\(\)](#), [vsnprintf\(\)](#), [vsprintf\(\)](#)

fpurge Function

- Purpose** Erases any input or output buffered in a stream.
- Declared In** `posix/stdio.h`
- Prototype** `int fpurge (FILE *stream)`
- Parameters**
- *stream*
The specified stream.
- Returns** Returns zero (0) upon successful completion. Otherwise, EOF is returned and the global variable `errno` is set to indicate the error.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).
- See Also** [fflush\(\)](#)

fputc Function

- Purpose** Writes a character (converted to an “unsigned char”) to an output stream.
- Declared In** `posix/stdio.h`
- Prototype** `int fputc (int c, FILE *stream)`
- Parameters**
- *c*
A character.
 - *stream*
The specified stream.
- Returns** Returns the character written. Otherwise, EOF is returned if an error occurs.
- Compatibility** This function *is* in the C99 specification.
- See Also** [getc\(\)](#), [putc\(\)](#)

fputs Function

- Purpose** Writes a line to a stream.
- Declared In** `posix/stdio.h`
- Prototype** `int fputs (const char *str, FILE *stream)`
- Parameters**
- *str*
A character string.
 - *stream*
The specified stream.
- Returns** Returns zero (0) upon successful completion and EOF on error.
- Compatibility** This function *is* in the C99 specification.
- See Also** [puts\(\)](#)

fread Function

- Purpose** Reads objects from the stream, storing them at the location specified by *ptr*.
- Declared In** `posix/stdio.h`
- Prototype** `size_t fread (void *ptr, size_t size, size_t nmemb, FILE *stream)`
- Parameters**
- *ptr*
The storage location.
 - *size*
The size of the object, in bytes.
 - *nmemb*
An object.
 - *stream*
The specified stream.
- Returns** Returns the number of objects read.
- Comments** Advances the file position indicator for the stream by the number of bytes read.
- Compatibility** This function *is* in the C99 specification.
- See Also** [read\(\)](#)

freopen Function

- Purpose** Opens the file whose name is the string pointed to by *path* and associates a stream with it.
- Declared In** `posix/stdio.h`
- Prototype** `FILE *freopen (const char *path, const char *mode, FILE *stream)`
- Parameters**
- *path*
A path pointing to a string containing a file name.
 - *mode*
A string indicating the mode.
 - *stream*
The specified stream.

- Returns** Returns a FILE pointer upon successful completion. Otherwise, NULL is returned and the global variable `errno` is set to indicate the error.
- Compatibility** This function *is* in the C99 specification.
- See Also** [fdopen\(\)](#), [fopen\(\)](#)

fscanf Function

- Purpose** Reads formatted input from a stream.
- Declared In** `posix/stdio.h`
- Prototype** `int fscanf (FILE *stream, const char *format, ...)`
- Parameters**
- *stream*
The specified stream.
 - *format*
The format string may contain conversion specifiers or other characters.
- Returns** Returns the number of input items assigned, which can be fewer than provided for, or even zero (0), in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned. The value EOF is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions that were successfully completed is returned.
- Comments** Scanning stops when an input character does not match such a format character. Scanning also stops when an input conversion cannot be made.
- Compatibility** This function *is* in the C99 specification.
- This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character. It also does not properly scan wide characters.
- See Also** [scanf\(\)](#), [sscanf\(\)](#), [vscanf\(\)](#), [vsscanf\(\)](#)

fseek Function

- Purpose** Sets the file position indicator for a stream. The new position, measured in bytes, is obtained by adding offset bytes to the position specified by *whence*.
- Declared In** `posix/stdio.h`
- Prototype** `int fseek (FILE *stream, long offset, int whence)`
- Parameters**
- *stream*
The specified stream.
 - *offset*
The number of bytes to add to the position.
 - *whence*
The position in a stream. If *whence* is set to `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.
- Returns** Returns zero (0) upon successful completion. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.
- Compatibility** This function *is* in the C99 specification.
- See Also** [fseeko\(\)](#), [ftell\(\)](#)

fseeko Function

- Purpose** Identical to the `fseek()` function except that the offset argument is of type `off_t`.
- Declared In** `posix/stdio.h`
- Prototype** `int fseeko (FILE *stream, off_t offset, int whence)`
- Parameters**
- *stream*
The specified stream.
 - *offset*
The number of bytes to add to the position.
 - *whence*
The position in a stream. If *whence* is set to `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, the offset is relative to the start of

the file, the current position indicator, or end-of-file, respectively.

Returns Returns zero (0) upon successful completion. Otherwise, a non-zero value is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

See Also [fseek\(\)](#)

fsetpos Function

Purpose Sets a file position for a stream.

Declared In `posix/stdio.h`

Prototype `int fsetpos (FILE *stream, const fpos_t *pos)`

Parameters `→ stream`
The specified stream.

`→ pos`
The current value of the file offset into the object.

Returns Returns zero (0) upon successful completion. Otherwise, a non-zero value is returned and the global variable `errno` is set to indicate the error.

Comments An alternative interface equivalent to `fseek()` and `fseeko()`.

Compatibility This function *is* in the C99 specification.

See Also [fgetpos\(\)](#)

ftell Function

Purpose Gets the current value of the file position indicator for a stream.

Declared In `posix/stdio.h`

Prototype `long ftell (FILE *stream)`

Parameters `→ stream`
The specified stream.

Returns Returns the current offset upon successful completion. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function *is* in the C99 specification.

See Also [fseek\(\)](#), [ftello\(\)](#)

ftello Function

Purpose Identical to the `ftell()` function except that the return value is of type `off_t`.

Declared In `posix/stdio.h`

Prototype `off_t ftello (FILE *stream)`

Parameters `→ stream`
The specified stream.

Returns Returns the current offset upon successful completion. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

See Also [ftell\(\)](#)

fwrite Function

Purpose Writes objects to the stream, obtaining them from the location specified by `ptr`.

Declared In `posix/stdio.h`

Prototype `size_t fwrite (const void *ptr, size_t size, size_t nmemb, FILE *stream)`

Parameters `→ ptr`
The storage location.
`→ size`
The size of the object, in bytes.
`→ nmemb`
An object.

→ *stream*
The specified stream.

- Returns** Returns the number of objects written.
- Comments** Advances the file position indicator for the stream by the number of bytes written.
- Compatibility** This function *is* in the C99 specification.
- See Also** [write\(\)](#)

getc Function

- Purpose** Gets a character from a stream.
- Declared In** `posix/stdio.h`
- Prototype** `int getc (FILE *stream)`
- Parameters** → *stream*
The specified stream.
- Returns** Returns the next requested object from the stream. Otherwise, EOF is returned if the stream is at end-of-file or a read error occurs.
- Comments** Essentially identical to `fgetc ()`, but is a macro that expands inline.
- Compatibility** This function *is* in the C99 specification.
- See Also** [fgetc\(\)](#), [putc\(\)](#)

getchar Function

- Purpose** Gets a character from the standard input stream `stdin`.
- Declared In** `posix/stdio.h`
- Prototype** `int getchar (void)`
- Returns** Returns the next requested object from the standard input stream `stdin`. Otherwise, EOF is returned if the stream is at end-of-file or a read error occurs.
- Comments** Identical to the `getc ()` function with the argument `stdin`.
- Compatibility** This function *is* in the C99 specification.
- See Also** [getc\(\)](#)

gets Function

Purpose	Gets a line from a stream.
Declared In	posix/stdio.h
Prototype	char *gets (char * <i>str</i>)
Parameters	→ <i>str</i> A character string.
Returns	Returns a pointer to the string upon successful completion. Otherwise, NULL is returned if the stream is at end-of-file or a read error occurs before any characters are read.
Comments	Identical to fgets () with an infinite size and a stream of stdin, except that the newline character (if any) is not stored in the string.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	fgets()

getw Function

Purpose	Gets the next int (if present) from a stream.
Declared In	posix/stdio.h
Prototype	int getw (FILE * <i>stream</i>)
Parameters	→ <i>stream</i> The specified stream.
Returns	Returns the next requested object from the stream. Otherwise, EOF is returned if the stream is at end-of-file or a read error occurs.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).
See Also	putw()

perror Function

Purpose	Writes an error to the standard error stream <code>stderr</code> .
Declared In	<code>posix/stdio.h</code>
Prototype	<code>void perror (const char *<i>string</i>)</code>
Parameters	<code>→ <i>string</i></code> The language-dependent error message string affiliated with an error number. If <i>string</i> is not <code>NULL</code> , <i>string</i> is prepended to the language-dependent error message string that is printed. That is, the message “<string>: <error string>” gets printed to <code>stderr</code> , where <error string> is the error message that corresponds to the error code found in the global variable <code>errno</code> .
Comments	The contents of the error message string is the same as those returned by <code>strerror()</code> with argument <code>errno</code> .
Compatibility	This function <i>is</i> in the C99 specification. This function is internationally safe to use, since it ultimately uses the <code>SysErrString()</code> function; see Exploring Palm OS: System Management .
See Also	strerror()

printf Function

Purpose	Writes formatted output to the standard output <code>stdout</code> .
Declared In	<code>posix/stdio.h</code>
Prototype	<code>int printf (const char *<i>format</i>, ...)</code>
Parameters	<code>→ <i>format</i></code> A string that specifies how subsequent arguments are converted for output.
Returns	Returns the number of characters transmitted, or a negative value if an output or encoding error occurred.
Compatibility	This function <i>is</i> in the C99 specification.

stdio.h

putc

This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.

See Also [asprintf\(\)](#), [fprintf\(\)](#), [snprintf\(\)](#), [sprintf\(\)](#), [vasprintf\(\)](#), [vfprintf\(\)](#), [vprintf\(\)](#), [vsnprintf\(\)](#), [vsprintf\(\)](#)

putc Function

Purpose Writes a character to a stream.

Declared In posix/stdio.h

Prototype int putc (int *c*, FILE **stream*)

Parameters

- *c*
A character.
- *stream*
The specified stream.

Returns Returns the character written. Otherwise, EOF is returned if an error occurs.

Comments Essentially identical to `fputc ()`, but is a macro that expands inline.

Compatibility This function *is* in the C99 specification.

See Also [fputc\(\)](#)

putchar Function

Purpose Writes a character to the standard output `stdout`.

Declared In posix/stdio.h

Prototype int putchar (int *c*)

Parameters

- *c*
A character.

Returns Returns the character written. Otherwise, EOF is returned if an error occurs.

Comments Identical to the `putc ()` function with the argument `stdout`.

Compatibility This function *is* in the C99 specification.

See Also [putc\(\)](#)

puts Function

Purpose Writes a string to the standard output `stdout`.

Declared In `posix/stdio.h`

Prototype `int puts (const char *str)`

Parameters `→ str`
A character string.

Returns Returns a non-negative integer upon successful completion and EOF on error.

Compatibility This function *is* in the C99 specification.

See Also [fputs\(\)](#)

putw Function

Purpose Writes the specified word to an output stream.

Declared In `posix/stdio.h`

Prototype `int putw (int w, FILE *stream)`

Parameters `→ w`
A word.
`→ stream`
The specified stream.

Returns Returns zero (0) upon successful completion. Otherwise, EOF is returned if a write error occurs, or if an attempt is made to write a read-only stream.

Compatibility This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

See Also [getw\(\)](#)

rewind Function

- Purpose** Resets the file indicator for a stream to the beginning.
- Declared In** `posix/stdio.h`
- Prototype** `void rewind (FILE *stream)`
- Parameters** \rightarrow *stream*
The specified stream.
- Compatibility** This function *is* in the C99 specification.

scanf Function

- Purpose** Reads formatted input from the standard input stream `stdin`.
- Declared In** `posix/stdio.h`
- Prototype** `int scanf (const char *format, ...)`
- Parameters** \rightarrow *format*
The format string may contain conversion specifiers or other characters.
- Returns** Returns the number of input items assigned, which can be fewer than provided for, or even zero (0), in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned. The value EOF is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions that were successfully completed is returned.
- Compatibility** This function *is* in the C99 specification.
This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character. It also does not properly scan wide characters.
- See Also** [fscanf\(\)](#), [sscanf\(\)](#), [vscanf\(\)](#), [vsscanf\(\)](#)

setbuf Function

Purpose	Sets the buffer size for a stream.
Declared In	posix/stdio.h
Prototype	void setbuf (FILE * <i>stream</i> , char * <i>buf</i>)
Parameters	→ <i>stream</i> The specified stream. → <i>buf</i> Points to a buffer.
Comments	An alias for calls to <code>setvbuf ()</code> .
Compatibility	This function <i>is</i> in the C99 specification.
See Also	setvbuf ()

setbuffer Function

Purpose	Sets the buffer size for a stream.
Declared In	posix/stdio.h
Prototype	void setbuffer (FILE * <i>stream</i> , char * <i>buf</i> , int <i>size</i>)
Parameters	→ <i>stream</i> The specified stream. → <i>buf</i> Points to a buffer at least <i>size</i> bytes long; this buffer is used instead of the current buffer. → <i>size</i> The size of the buffer.
Comments	An alias for calls to <code>setvbuf ()</code> .
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	setvbuf ()

setlinebuf Function

Purpose	Sets the buffer size for a stream.
Declared In	posix/stdio.h
Prototype	<code>int setlinebuf (FILE *<i>stream</i>)</code>
Parameters	→ <i>stream</i> The specified stream.
Returns	Returns what the equivalent <code>setvbuf ()</code> would have returned.
Comments	An alias for calls to <code>setvbuf ()</code> .
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).
See Also	setvbuf ()

setvbuf Function

Purpose	Sets the buffer size and scheme for a stream. Used to alter the buffering behavior of a stream.
Declared In	posix/stdio.h
Prototype	<code>int setvbuf (FILE *<i>stream</i>, char *<i>buf</i>, int <i>mode</i>, size_t <i>size</i>)</code>
Parameters	→ <i>stream</i> The specified stream. → <i>buf</i> A buffer. → <i>mode</i> Must be one of the following: <code>_IONBF</code> , <code>_IOLBF</code> , or <code>IOFBF</code> , which represents the three types of buffering available (unbuffered, line buffered, or fully buffered, respectively). → <i>size</i> May be specified as zero (0) to obtain deferred optimal-size buffer allocation as usual. If it is not zero, then except for unbuffered files, <i>buf</i> should point to a buffer at least <i>size</i> bytes long; this buffer is used instead of the current buffer.

- Returns** Returns zero (0) upon successful completion, or EOF if the request cannot be honored.
- Compatibility** This function *is* in the C99 specification.
- See Also** [setbuf\(\)](#), [setbuffer\(\)](#), [setlinebuf\(\)](#)

snprintf Function

- Purpose** Writes formatted output to a character string.
- Declared In** posix/stdio.h
- Prototype**

```
int snprintf (char *str, size_t size,
              const char *format, ...)
```
- Parameters**
- *str*
A character string.
 - *size*
The number of characters.
 - *format*
A string that specifies how subsequent arguments are converted for output.
- Returns** Returns the number of characters that would have been written had *size* been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred.
- Comments** Writes at most *size*-1 of the characters printed into the output string (the *size*th character then gets the terminating '\0') if the return value is greater than or equal to *size*, the string was too short and some of the printed characters were discarded. If *size* is zero (0), nothing is written and *str* may be a NULL pointer.
- Compatibility** This function *is* in the C99 specification.
- This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.
- See Also** [asprintf\(\)](#), [fprintf\(\)](#), [printf\(\)](#), [sprintf\(\)](#), [vasprintf\(\)](#), [vfprintf\(\)](#), [vprintf\(\)](#), [vsnprintf\(\)](#), [vsprintf\(\)](#)

sprintf Function

Purpose	Writes formatted output to a character string.
Declared In	posix/stdio.h
Prototype	<code>int sprintf (char *<i>str</i>, const char *<i>format</i>, ...)</code>
Parameters	<p>→ <i>str</i> A character string.</p> <p>→ <i>format</i> A string that specifies how subsequent arguments are converted for output.</p>
Returns	Returns the number of characters written in the array, not counting the terminating null character, or a negative value if an encoding error occurred.
Comments	Effectively assumes an infinite size.
Compatibility	This function <i>is</i> in the C99 specification. This function is internationally safe to use <i>except</i> for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.
See Also	asprintf() , fprintf() , printf() , snprintf() , vasprintf() , vfprintf() , vprintf() , vsnprintf() , vsprintf()

sscanf Function

Purpose	Reads formatted input from a character string.
Declared In	posix/stdio.h
Prototype	<code>int sscanf (const char *<i>str</i>, const char *<i>format</i>, ...)</code>
Parameters	<p>→ <i>str</i> A character string.</p> <p>→ <i>format</i> The format string may contain conversion specifiers or other characters.</p>
Returns	Returns the number of input items assigned, which can be fewer than provided for, or even zero (0), in the event of a matching

failure. Zero indicates that, while there was input available, no conversions were assigned. The value EOF is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions that were successfully completed is returned.

Compatibility This function *is* in the C99 specification.

This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character. It also does not properly scan wide characters.

See Also [fscanf\(\)](#), [scanf\(\)](#), [vscanf\(\)](#), [vsscanf\(\)](#)

ungetc Function

Purpose Places a character back in a stream.

Declared In `posix/stdio.h`

Prototype `int ungetc (int c, FILE *stream)`

Parameters `→ c`

A character.

`→ stream`

The specified stream.

Returns Returns the character pushed-back after the conversion, or EOF if the operation fails.

Compatibility This function *is* in the C99 specification.

See Also [fgetc\(\)](#)

vasprintf Function

Purpose Writes to a dynamically allocated string that is stored in *ret*.

Declared In `posix/stdio.h`

Prototype `int vasprintf (char **ret, const char *format, va_list ap)`

stdio.h

vfprintf

- Parameters**
- *ret*
A dynamically allocated string.
 - *format*
A string that specifies how subsequent arguments are converted for output.
 - *ap*
An object of type `va_list` initialized by `va_start()`.
- Returns** Returns a pointer to a buffer sufficiently large to hold the string in the *ret* argument.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).
This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.
- See Also** [asprintf\(\)](#), [fprintf\(\)](#), [printf\(\)](#), [snprintf\(\)](#), [sprintf\(\)](#), [vfprintf\(\)](#), [vprintf\(\)](#), [vsnprintf\(\)](#), [vsprintf\(\)](#)

vfprintf Function

- Purpose** Writes formatted output to an output stream using an argument list.
- Declared In** `posix/stdio.h`
- Prototype** `int vfprintf (FILE *stream, const char *format, va_list ap)`
- Parameters**
- *stream*
The specified stream.
 - *format*
A string that specifies how subsequent arguments are converted for output.
 - *ap*
An object of type `va_list` initialized by `va_start()`.
- Returns** Returns the number of characters transmitted, or a negative value if an output or encoding error occurred.
- Compatibility** This function *is* in the C99 specification.

This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.

See Also [asprintf\(\)](#), [fprintf\(\)](#), [printf\(\)](#), [snprintf\(\)](#), [sprintf\(\)](#), [vasprintf\(\)](#), [vprintf\(\)](#), [vsnprintf\(\)](#), [vsprintf\(\)](#)

vprintf Function

Purpose Writes formatted output to the standard output `stdout` using an argument list.

Declared In `posix/stdio.h`

Prototype `int vprintf (const char *format, va_list ap)`

Parameters → *format*

A string that specifies how subsequent arguments are converted for output.

→ *ap*

An object of type `va_list` initialized by `va_start()`.

Returns Returns the number of characters transmitted, or a negative value if an output or encoding error occurred.

Compatibility This function *is* in the C99 specification.

This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.

See Also [asprintf\(\)](#), [fprintf\(\)](#), [printf\(\)](#), [snprintf\(\)](#), [sprintf\(\)](#), [vasprintf\(\)](#), [vfprintf\(\)](#), [vsnprintf\(\)](#), [vsprintf\(\)](#)

vscanf Function

Purpose Reads formatted input from the standard output `stdout` using an argument list.

Declared In `posix/stdio.h`

Prototype `int vscanf (const char *format, va_list ap)`

stdio.h

vsnprintf

Parameters → *format*
The format string may contain conversion specifiers or other characters.

→ *ap*
An object of type `va_list` initialized by `va_start()`.

Returns Returns the number of input items assigned, which can be fewer than provided for, or even zero (0), in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned. The value EOF is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions that were successfully completed is returned.

Compatibility This function *is* in the C99 specification.

This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character. It also does not properly scan wide characters.

See Also [fscanf\(\)](#), [scanf\(\)](#), [sscanf\(\)](#), [vsscanf\(\)](#)

vsnprintf Function

Purpose Writes formatted output to a character string.

Declared In `posix/stdio.h`

Prototype `int vsnprintf (char *str, size_t size,
const char *format, va_list ap)`

Parameters → *str*
A character string.

→ *size*
The number of characters.

→ *format*
A string that specifies how subsequent arguments are converted for output.

- *ap*
An object of type `va_list` initialized by `va_start()`.
- Returns** Returns the number of characters that would have been written had *size* been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred.
- Comments** Writes at most *size*-1 of the characters printed into the output string (the *size*th character then gets the terminating '\0') if the return value is greater than or equal to *size*, the string was too short and some of the printed characters were discarded. If *size* is zero (0), nothing is written and *str* may be a NULL pointer.
- Compatibility** This function *is* in the C99 specification.
This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.
- See Also** [asprintf\(\)](#), [fprintf\(\)](#), [printf\(\)](#), [snprintf\(\)](#), [sprintf\(\)](#), [vasprintf\(\)](#), [vfprintf\(\)](#), [vprintf\(\)](#), [vsprintf\(\)](#)

vsprintf Function

- Purpose** Writes formatted output to a string using an argument list.
- Declared In** `posix/stdio.h`
- Prototype** `int vsprintf (char *str, const char *format, va_list ap)`
- Parameters**
- *str*
A character string.
- *format*
A string that specifies how subsequent arguments are converted for output.
- *ap*
An object of type `va_list` initialized by `va_start()`.
- Returns** Returns the number of characters written in the array, not counting the terminating null character, or a negative value if an encoding error occurred.
- Comments** Effectively assumes an infinite size.

Compatibility This function *is* in the C99 specification.
This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character.

See Also [asprintf\(\)](#), [fprintf\(\)](#), [printf\(\)](#), [snprintf\(\)](#), [sprintf\(\)](#), [vasprintf\(\)](#), [vfprintf\(\)](#), [vprintf\(\)](#), [vsnprintf\(\)](#)

vsscanf Function

Purpose Reads formatted input from a string using an argument list.

Declared In `posix/stdio.h`

Prototype `int vsscanf (const char *str, const char *format, va_list ap)`

Parameters

- *str*
A character string.
- *format*
The format string may contain conversion specifiers or other characters.
- *ap*
An object of type `va_list` initialized by `va_start()`.

Returns Returns the number of input items assigned, which can be fewer than provided for, or even zero (0), in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned. The value EOF is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions that were successfully completed is returned.

Compatibility This function *is* in the C99 specification.

This function is internationally safe to use *except* for formatting floating point numbers, since it does not use a locale-sensitive decimal point character. It also does not properly scan wide characters.

See Also [fscanf\(\)](#), [scanf\(\)](#), [sscanf\(\)](#), [vscanf\(\)](#)

stdlib.h

The <stdlib.h> header defines several general operation functions and macros.

Structures and Types

div_t Struct

Purpose	The structure returned by the <code>div</code> function.
Declared In	<code>posix/stdlib.h</code>
Prototype	<pre>typedef struct { int quot; int rem; } div_t</pre>
Fields	<code>quot</code> The quotient. <code>rem</code> The remainder.

ldiv_t Struct

Purpose	The structure returned by the <code>ldiv</code> function.
Declared In	<code>posix/stdlib.h</code>
Prototype	<pre>typedef struct { long quot; long rem; } ldiv_t</pre>
Fields	<code>quot</code> The quotient. <code>rem</code> The remainder.

lldiv_t Struct

Purpose	The structure returned by the <code>lldiv</code> function.				
Declared In	<code>posix/stdlib.h</code>				
Prototype	<pre>typedef struct { int64_t quot; int64_t rem; } lldiv_t</pre>				
Fields	<table><tr><td><code>quot</code></td><td>The quotient.</td></tr><tr><td><code>rem</code></td><td>The remainder.</td></tr></table>	<code>quot</code>	The quotient.	<code>rem</code>	The remainder.
<code>quot</code>	The quotient.				
<code>rem</code>	The remainder.				

qdiv_t Struct

Purpose	The structure returned by the <code>qdiv</code> function.				
Declared In	<code>posix/stdlib.h</code>				
Prototype	<pre>typedef struct { quad_t quot; quad_t rem; } qdiv_t</pre>				
Fields	<table><tr><td><code>quot</code></td><td>The quotient.</td></tr><tr><td><code>rem</code></td><td>The remainder.</td></tr></table>	<code>quot</code>	The quotient.	<code>rem</code>	The remainder.
<code>quot</code>	The quotient.				
<code>rem</code>	The remainder.				

Functions and Macros

abs Function

Purpose	Computes the absolute value of an integer.
Declared In	posix/stdlib.h
Prototype	int abs (int <i>j</i>)
Parameters	→ <i>j</i> An integer.
Returns	Returns the absolute value.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	labs() , llabs()

atof Function

Purpose	Converts a character string to a numeric value of type double.
Declared In	posix/stdlib.h
Prototype	double atof (const char * <i>nptr</i>)
Parameters	→ <i>nptr</i> The string to be converted to a floating-point number.
Returns	Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use because it is not multi-byte aware. It also does not use a locale-sensitive decimal point character for formatting floating point numbers.
See Also	atoi() , atol() , atoll()

atoi Function

Purpose	Converts a character string to a numeric value of type <code>int</code> .
Declared In	<code>posix/stdlib.h</code>
Prototype	<code>int atoi (const char *nptr)</code>
Parameters	<code>→ nptr</code> The string to be converted to an integer.
Returns	Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use because it is not multi-byte aware. The Palm OS equivalent of this function is the <code>StrAToI()</code> function; see Exploring Palm OS: Text and Localization .
See Also	atof() , atol() , atoll()

atol Function

Purpose	Converts a character string to a numeric value of type <code>long</code> .
Declared In	<code>posix/stdlib.h</code>
Prototype	<code>long atol (const char *nptr)</code>
Parameters	<code>→ nptr</code> The string to be converted to a long integer.
Returns	Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use because it is not multi-byte aware. The Palm OS equivalent of this function is the <code>StrAToI()</code> function; see Exploring Palm OS: Text and Localization .
See Also	atof() , atoi() , atoll()

atoll Function

- Purpose** Converts a character string to a numeric value of type long long.
- Declared In** posix/stdlib.h
- Prototype** int64_t atoll (const char *nptr)
- Parameters** → *nptr*
The string to be converted to a long long integer.
- Returns** Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware.
- See Also** [atof\(\)](#), [atoi\(\)](#), [atol\(\)](#)

bsearch Function

- Purpose** Performs a binary search.
- Declared In** posix/stdlib.h
- Prototype** void *bsearch (const void *key, const void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *))
- Parameters** → *key*
An element of the array.
- *base*
The beginning of the array.
- *nmemb*
The number of members in the array.
- *size*
The size of each element in the array, specified in bytes.
- *compar*
The `compar ()` function. This function takes two arguments, the first is the key pointer and the second is the current element in the array being compared.

- Returns** Returns a pointer to a match, if a match is found. Otherwise, a NULL pointer is returned.
- Compatibility** This function *is* in the C99 specification.
- See Also** [qsort\(\)](#), [qsort_r\(\)](#)

calloc Function

- Purpose** Allocates space for a group of objects.
- Declared In** `posix/stdlib.h`
- Prototype** `void *calloc (size_t number, size_t size)`
- Parameters**
- *number*
The number of objects to allocate space for.
 - *size*
The length in bytes of each object to allocate space for.
- Returns** Returns a pointer to the allocated memory upon successful completion. Otherwise, a NULL pointer is returned.
- Compatibility** This function *is* in the C99 specification.
- See Also** [free\(\)](#), [malloc\(\)](#), [realloc\(\)](#)

div Function

- Purpose** Divides the numerator by the denominator.
- Declared In** `posix/stdlib.h`
- Prototype** `div_t div (int num, int denom)`
- Parameters**
- *num*
The numerator.
 - *denom*
The denominator.
- Returns** Returns the quotient and remainder in a `div_t` structure.
- Compatibility** This function *is* in the C99 specification.
- See Also** [ldiv\(\)](#)

free Function

Purpose	Releases previously allocated memory to heap.
Declared In	posix/stdlib.h
Prototype	void free (void * <i>ptr</i>)
Parameters	→ <i>ptr</i> The allocated memory to free.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).
See Also	calloc() , malloc() , realloc()

getenv Function

Purpose	Gets the current value of an environment variable.
Declared In	posix/stdlib.h
Prototype	char *getenv (const char * <i>name</i>)
Parameters	→ <i>name</i> The environment variable.
Returns	Returns a pointer to the current value upon successful completion. Otherwise, a NULL pointer is returned if <i>name</i> is not in the current environment.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	putenv() , setenv()

inplace_realloc Function

Purpose	Attempts to resize the memory block inplace.
Declared In	posix/stdlib.h
Prototype	void *inplace_realloc (void * <i>ptr</i> , size_t <i>size</i>)
Parameters	→ <i>ptr</i> The previously allocated memory. → <i>size</i> The size, in bytes, to change to.

stdlib.h

labs

Returns Returns a pointer, possibly identical to *ptr*, to the allocated memory upon successful completion. Otherwise, a NULL pointer is returned, in which case the memory referenced by *ptr* is still available and intact.

Compatibility This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

See Also [realloc\(\)](#)

labs Function

Purpose Computes the long integer absolute value.

Declared In `posix/stdlib.h`

Prototype `long labs (long j)`

Parameters $\rightarrow j$
A value of type long.

Returns Returns the absolute value.

Compatibility This function *is* in the C99 specification.

See Also [abs\(\)](#), [llabs\(\)](#)

ldiv Function

Purpose Divides the numerator by the denominator.

Declared In `posix/stdlib.h`

Prototype `ldiv_t ldiv (long num, long denom)`

Parameters $\rightarrow num$
The numerator.
 $\rightarrow denom$
The denominator.

Returns Returns a long integer value.

Compatibility This function *is* in the C99 specification.

See Also [div\(\)](#)

llabs Function

Purpose	Computes the long long integer absolute value.
Declared In	posix/stdlib.h
Prototype	<code>int64_t llabs (int64_t j)</code>
Parameters	<code>→ j</code> A value of type <code>long long</code> .
Returns	Returns the absolute value.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	abs() , labs()

malloc Function

Purpose	Allocates a block of memory heap.
Declared In	posix/stdlib.h
Prototype	<code>void *malloc (size_t size)</code>
Parameters	<code>→ size</code> The bytes of memory to allocate.
Returns	Returns a pointer to the allocated memory upon successful completion. Otherwise, a NULL pointer is returned.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	calloc() , free() , realloc()

putenv Function

Purpose	Enters an argument into the environment list.
Declared In	posix/stdlib.h
Prototype	<code>int putenv (const char *string)</code>
Parameters	<code>→ string</code> The item to add to the environment list.
Returns	Returns zero (0) upon successful completion. Otherwise, -1 is returned and the global variable <code>errno</code> is set to indicate the error.

Compatibility This function *is* in the C99 specification.

See Also [getenv\(\)](#), [setenv\(\)](#)

qsort Function

Purpose Sorts an array.

Declared In posix/stdlib.h

Prototype
`void qsort (void *base, size_t nmemb,
 size_t size, int (*compar)(const void *,
 const void *))`

Parameters

- *base*
The beginning of the array.
- *nmemb*
The number of members in the array.
- *size*
The size of each element in the array, specified in bytes.
- *compar*
The `compar()` function. This function takes two arguments to be compared. The elements are sorted in ascending order.

Compatibility This function *is* in the C99 specification.

See Also [bsearch\(\)](#), [qsort_r\(\)](#)

qsort_r Function

Purpose Re-entrant interface to `qsort()`, which sorts an array.

Declared In posix/stdlib.h

Prototype
`void qsort_r (void *base, size_t nmemb,
 size_t size, void *cookie,
 int (*compar)(void *, const void *,
 const void *))`

Parameters

- *base*
The beginning of the array.

→ *nmemb*

The number of members in the array.

→ *size*

The size of each element in the array, specified in bytes.

→ *cookie*

An argument that is passed unchanged as the first argument to the function pointed to by *compar*. This allows the comparison function to access additional data without using global variables.

→ *compar*

The `compar ()` function. This function takes two arguments to be compared. The elements are sorted in ascending order.

Compatibility This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

See Also [bsearch\(\)](#), [qsort\(\)](#)

rand Function

Purpose Generates a pseudo-random integer value.

Declared In `posix/stdlib.h`

Prototype `int rand (void)`

Returns Returns a pseudo-random integer value.

Compatibility This function *is* in the C99 specification.

See Also [rand_r\(\)](#), [random\(\)](#), [srand\(\)](#)

rand_r Function

Purpose Re-entrant interface to `rand ()`, which generates a pseudo-random integer value.

Declared In `posix/stdlib.h`

Prototype `int rand_r (unsigned int *seed)`

Parameters → *seed*
The user-provided seed.

- Returns** Returns a pseudo-random integer value.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [rand\(\)](#)

random Function

- Purpose** Uses a random number generator to return successive pseudo-random numbers in the range from zero (0) to $(2^{31})-1$.
- Declared In** `posix/stdlib.h`
- Prototype** `long random (void)`
- Returns** Returns a random number in the range from zero (0) to $(2^{31})-1$.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [rand\(\)](#), [srandom\(\)](#)

realloc Function

- Purpose** Changes the size of an allocated block of heap memory.
- Declared In** `posix/stdlib.h`
- Prototype** `void *realloc (void *ptr, size_t size)`
- Parameters**
- `→ ptr`
The previously allocated memory.
 - `→ size`
The size in bytes to change to.
- Returns** Returns a pointer, possibly identical to `ptr`, to the allocated memory upon successful completion. Otherwise, a NULL pointer is returned, in which case the memory referenced by `ptr` is still available and intact.
- Compatibility** This function *is* in the C99 specification.
- See Also** [calloc\(\)](#), [free\(\)](#), [malloc\(\)](#)

setenv Function

Purpose	Inserts or resets an environment variable in the current environment list.
Declared In	posix/stdlib.h
Prototype	<pre>int setenv (const char *name, const char *value, int overwrite)</pre>
Parameters	<p>→ <i>name</i> The environment variable.</p> <p>→ <i>value</i> If <i>name</i> does not exist in the list, it is inserted with the specified <i>value</i>.</p> <p>→ <i>overwrite</i> If <i>name</i> does exist, the argument <i>overwrite</i> is tested. If <i>overwrite</i> is zero (0), <i>name</i> is not reset, otherwise it is reset to the specified <i>value</i>.</p>
Returns	Returns zero (0) upon successful completion. Otherwise, -1 is returned and the global variable <code>errno</code> is set to indicate the error.
Comments	This function does not actually do anything outside of the BCommand environment.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	getenv() , putenv() , unsetenv()

srand Function

Purpose	Seeds the random number generator used by <code>rand()</code> .
Declared In	posix/stdlib.h
Prototype	<pre>void srand (unsigned seed)</pre>
Parameters	<p>→ <i>seed</i> If no seed value is provided, the <code>rand()</code> function is automatically seeded with a value of 1.</p>
Compatibility	This function <i>is</i> in the C99 specification.
See Also	rand() , srandom()

srandom Function

- Purpose** Seeds the random number generator used by `random()`.
- Declared In** `posix/stdlib.h`
- Prototype** `void srandom (unsigned long seed)`
- Parameters** \rightarrow *seed*
If no seed value is provided, the `random()` function is automatically seeded with a value of 1.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [random\(\)](#), [srand\(\)](#)

strtod Function

- Purpose** Converts a character array to a floating-point value of type `double`.
- Declared In** `posix/stdlib.h`
- Prototype** `double strtod (const char *nptr, char **endptr)`
- Parameters** \rightarrow *nptr*
The string to be converted to a floating-point number.
- \rightarrow *endptr*
A pointer to a pointer. The address of the first invalid character is stored in the pointer that *endptr* points to.
- Returns** Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
- Compatibility** This function *is* in the C99 specification.

This function is *not* internationally safe to use because it is not multi-byte aware. It also does not use a locale-sensitive decimal point character for formatting floating point numbers.
- See Also** [strtol\(\)](#), [strtoll\(\)](#), [strtoul\(\)](#), [strtoull\(\)](#)

strtol Function

Purpose	Converts a character array to an integral value of type long int.
Declared In	posix/stdlib.h
Prototype	long strtol (const char *nptr, char **endptr, int base)
Parameters	→ <i>nptr</i> The string to be converted to a long integer. → <i>endptr</i> A pointer to a pointer. The address of the first invalid character is stored in the pointer that <i>endptr</i> points to. → <i>base</i> The radix.
Returns	Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use because it is not multi-byte aware. The Palm OS equivalent of this function is the <code>strAToI()</code> function; see Exploring Palm OS: Text and Localization .
See Also	strtod() , strtoll() , strtoul() , strtoull()

strtoll Function

Purpose	Converts a character array to an integer value of type long long int.
Declared In	posix/stdlib.h
Prototype	int64_t strtoll (const char *nptr, char **endptr, int base)
Parameters	→ <i>nptr</i> The string to be converted to a long long integer. → <i>endptr</i> A pointer to a pointer. The address of the first invalid character is stored in the pointer that <i>endptr</i> points to. → <i>base</i> The radix.

stdlib.h

strtoul

- Returns** Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware.
- See Also** [strtod\(\)](#), [strtol\(\)](#), [strtoul\(\)](#), [strtoull\(\)](#)

strtoul Function

- Purpose** Converts a character array to an integer value of type unsigned long.
- Declared In** posix/stdlib.h
- Prototype** unsigned long strtoul (const char *nptr,
char **endptr, int base)
- Parameters**
- *nptr*
The string to be converted to an unsigned long integer.
 - *endptr*
A pointer to a pointer. The address of the first invalid character is stored in the pointer that *endptr* points to.
 - *base*
The radix.
- Returns** Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware.
- See Also** [strtod\(\)](#), [strtol\(\)](#), [strtoll\(\)](#), [strtoull\(\)](#)

strtoull Function

Purpose	Converts a character array to an integer value of type unsigned long long int.
Declared In	posix/stdlib.h
Prototype	<pre>uint64_t strtoull (const char *nptr, char **endptr, int base)</pre>
Parameters	<p>→ <i>nptr</i> The string to be converted to an unsigned long long integer.</p> <p>→ <i>endptr</i> A pointer to a pointer. The address of the first invalid character is stored in the pointer that <i>endptr</i> points to.</p> <p>→ <i>base</i> The radix.</p>
Returns	Returns the converted number upon successful completion. Otherwise, zero (0) is returned if no conversion can be made.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use because it is not multi-byte aware.
See Also	strtod() , strtol() , strtoll() , strtol()

unsetenv Function

Purpose	Deletes all instances of an environment variable from the environment list.
Declared In	posix/stdlib.h
Prototype	<pre>void unsetenv (const char *name)</pre>
Parameters	<p>→ <i>name</i> The environment variable.</p>
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	setenv()

stdlib.h

unsetenv

string.h

The `<string.h>` header defines several functions useful for manipulating strings (character arrays).

Functions and Macros

memchr Function

Purpose	Searches for an occurrence of a byte in a buffer.
Declared In	<code>posix/string.h</code>
Prototype	<code>void *memchr (const void *b, int c, size_t len)</code>
Parameters	<code>→ b</code> The buffer to search. <code>→ c</code> The byte to search for. <code>→ len</code> The length of bytes to search in.
Returns	Returns a pointer to the byte located, or a NULL pointer if no such byte exists within <code>len</code> bytes.
Compatibility	This function <i>is</i> in the C99 specification.

memcmp Function

Purpose	Compares two blocks of memory.
Declared In	<code>posix/string.h</code>
Prototype	<code>int memcmp (const void *b1, const void *b2, size_t len)</code>

string.h

memcpy

- Parameters**
- *b1*
A pointer to the first buffer of bytes to compare.
 - *b2*
A pointer to the second buffer of bytes to compare.
 - *len*
The length of each buffer in bytes.
- Returns** Returns zero (0) if the two buffers are identical. Otherwise, the difference between the first two differing bytes is returned.
- Compatibility** This function *is* in the C99 specification.

memcpy Function

- Purpose** Copies a contiguous memory block.
- Declared In** posix/string.h
- Prototype** void *memcpy (void **dst*, const void **src*, size_t *len*)
- Parameters**
- *dst*
A pointer to the destination buffer of bytes.
 - *src*
A pointer to the source buffer of bytes.
 - *len*
The length of bytes to copy to the specified buffer.
- Returns** Returns a pointer to the original value of *dst*.
- Compatibility** This function *is* in the C99 specification.
- See Also** [memmove\(\)](#)

memmove Function

- Purpose** Copies a contiguous memory block.
- Declared In** posix/string.h
- Prototype** void *memmove (void **dst*, const void **src*, size_t *len*)

Parameters	<p>→ <i>dst</i> A pointer to the destination buffer of bytes.</p> <p>→ <i>src</i> A pointer to the source buffer of bytes.</p> <p>→ <i>len</i> The length of bytes to copy to the specified buffer.</p>
Returns	Returns a pointer to the original value of <i>dst</i> .
Compatibility	<p>This function <i>is</i> in the C99 specification.</p> <p>The Palm OS equivalent of this function is the MemMove () function; see Exploring Palm OS: Memory, Databases, and Files. The MemMove () function is provided for backward compatibility.</p>
See Also	memcpy ()

memset Function

Purpose	Copies the value of <i>c</i> (the least significant byte) into each of the first <i>len</i> bytes of the buffer <i>b</i> .
Declared In	posix/string.h
Prototype	void *memset (void * <i>b</i> , int <i>c</i> , size_t <i>len</i>)
Parameters	<p>→ <i>b</i> The buffer to write to.</p> <p>→ <i>c</i> The byte to write.</p> <p>→ <i>len</i> The length of bytes to write to the specified buffer.</p>
Returns	Returns a pointer to the original value of <i>b</i> .
Compatibility	<p>This function <i>is</i> in the C99 specification; however, the parameters are different.</p> <p>The Palm OS equivalent of this function is the MemSet () function; see Exploring Palm OS: Memory, Databases, and Files. The MemSet () function is provided for backward compatibility. Note that the MemSet () function reverses the meaning of the last two parameters.</p>

strcat Function

Purpose	Concatenates two strings.
Declared In	posix/string.h
Prototype	char *strcat (char *s, const char *append)
Parameters	<p>→ <i>s</i> The null-terminated string to append to.</p> <p>→ <i>append</i> The null-terminated string to append.</p>
Returns	Returns a pointer to the concatenated string.
Compatibility	<p>This function <i>is</i> in the C99 specification.</p> <p>This function is internationally safe to use.</p> <p>The Palm OS equivalent of this function is the StrCat () function; see Exploring Palm OS: Text and Localization. The StrCat () function is provided for backward compatibility.</p>
See Also	strncat()

strchr Function

Purpose	Searches for the first occurrence of a character in a string.
Declared In	posix/string.h
Prototype	char *strchr (const char *s, int c)
Parameters	<p>→ <i>b</i> The string to search.</p> <p>→ <i>c</i> The character to search for.</p>
Returns	Returns a pointer to the located character, or a NULL pointer if the character does not appear in the string.
Compatibility	<p>This function <i>is</i> in the C99 specification.</p> <p>This function is <i>not</i> internationally safe to use because it is not multi-byte aware. The Palm OS equivalent of this function is the StrChr () function; see Exploring Palm OS: Text and Localization.</p>
See Also	strrchr()

strcmp Function

Purpose	Compares two strings.
Declared In	posix/string.h
Prototype	int strcmp (const char *s1, const char *s2)
Parameters	→ s1 The first string to compare. → s2 The second string to compare.
Returns	Returns an integer greater than, equal to, or less than zero (0), accordingly as the string s1 is greater than, equal to, or less than the string s2.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use because it is not multi-byte aware and not locale sensitive. The Palm OS equivalent of this function is the StrCompare () function; see Exploring Palm OS: Text and Localization .
See Also	strncmp ()

strcoll Function

Purpose	Compares two strings according to locale.
Declared In	posix/string.h
Prototype	int strcoll (const char *s1, const char *s2)
Parameters	→ s1 The first string to compare. → s2 The second string to compare.
Returns	Returns an integer greater than, equal to, or less than zero (0), accordingly as the string s1 is greater than, equal to, or less than the string s2.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use because it is not multi-byte aware and not locale sensitive. The Palm OS equivalent

string.h

strcpy

of this function is the `StrCompare()` function; see [Exploring Palm OS: Text and Localization](#).

strcpy Function

Purpose	Copies one string to another.
Declared In	posix/string.h
Prototype	<code>char *strcpy (char *dst, const char *src)</code>
Parameters	<p>→ <i>dst</i> The destination string.</p> <p>→ <i>src</i> The source string.</p>
Returns	Returns a pointer to the destination string.
Compatibility	<p>This function <i>is</i> in the C99 specification.</p> <p>This function is internationally safe to use.</p> <p>The Palm OS equivalent of this function is the <code>StrCopy()</code> function; see Exploring Palm OS: Text and Localization. The <code>StrCopy()</code> function is provided for backward compatibility.</p>
See Also	strncpy()

strcspn Function

Purpose	Finds the first sequence of characters in the string <i>s</i> that does not contain any character specified in <i>charset</i> .
Declared In	posix/string.h
Prototype	<code>size_t strcspn (const char *s, const char *charset)</code>
Parameters	<p>→ <i>s</i> The string to span.</p> <p>→ <i>charset</i> The string of characters to search for.</p>
Returns	Returns the length of this first sequence of characters found that do not match with <i>charset</i> .

Compatibility This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware.

See Also [strspn\(\)](#)

strdup Function

Purpose Saves a copy of a string.

Declared In posix/string.h

Prototype char *strdup (const char **str*)

Parameters → *str*
The string to copy.

Returns Returns a pointer to the copied string. Otherwise, a NULL pointer is returned if insufficient memory is available.

Compatibility This function *is* in the C99 specification.
This function is internationally safe to use.

strerror Function

Purpose Translates an error number into an error message.

Declared In posix/string.h

Prototype char *strerror (int *errnum*)

Parameters → *errnum*
The error number.

Returns Returns a pointer to the language-dependent error message string associated with the error number.

Compatibility This function is *not* in the C99 specification.
This function is internationally safe to use.

The Palm OS equivalent of this function is the SysErrString() function; see [Exploring Palm OS: System Management](#). The SysErrString() function is provided for backward compatibility.

See Also [strerror_r\(\)](#)

strerror_r Function

- Purpose** Re-entrant interface to `strerror()`, which translates an error number into an error message.
- Declared In** `posix/string.h`
- Prototype**
`char *strerror_r (int errnum, char *buf,
size_t buflen)`
- Parameters**
- *errnum*
The error number.
 - *buf*
The user-provided buffer for the resulting error message.
 - *buflen*
The length of the buffer. Note that this is one byte less than the size of the buffer in bytes.
- Returns** Returns a pointer to the language-dependent error message string associated with the error number.
- Compatibility** This function *is* in the C99 specification.
This function is internationally safe to use.
The Palm OS equivalent of this function is the `SysErrString()` function; see [Exploring Palm OS: System Management](#). The `SysErrString()` function is provided for backward compatibility.
- See Also** [strerror\(\)](#)

strlcat Function

- Purpose** Concatenates the null-terminated string *src* to the end of *dst*.
- Declared In** `posix/string.h`
- Prototype**
`size_t strlcat (char *dst, const char *src,
size_t size)`
- Parameters**
- *dst*
The destination string.
 - *src*
The source string.

→ *size*

The full size of the buffer to copy.

Returns Returns the total length of the string the function tried to create.

Comments Takes the full size of the buffer (not just the length) and guarantees to null-terminate the result (as long as *size* is larger than zero (0) or, as long as there is at least one byte free in *dst*). Appends at most *size - strlen(dst) - 1* bytes.

Compatibility This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

This function is *not* internationally safe to use because it is not multi-byte aware. The Palm OS equivalent of this function is the `StrLCat()` function; see [Exploring Palm OS: Text and Localization](#).

See Also [strncpy\(\)](#)

strncpy Function

Purpose Copies up to *size - 1* characters from the null-terminated string *src* to *dst*, null-terminating the result.

Declared In `posix/string.h`

Prototype `size_t strncpy (char *dst, const char *src,
size_t copy)`

Parameters → *dst*

The destination string.

→ *src*

The source string.

→ *size*

The full size of the buffer to copy.

Returns Returns the total length of the string the function tried to create.

Comments Takes the full size of the buffer (not just the length) and guarantees to null-terminate the result (as long as *size* is larger than zero (0)).

Compatibility This function is *not* in the C99 specification.

This function *is* a Palm OS extension (not present in C99 or Unix).

string.h

strlen

This function is *not* internationally safe to use because it is not multi-byte aware. The Palm OS equivalent of this function is the `StrLCopy()` function; see [Exploring Palm OS: Text and Localization](#).

See Also [strlcat\(\)](#)

strlen Function

Purpose Computes the length of a string.

Declared In `posix/string.h`

Prototype `size_t strlen (const char *s)`

Parameters `→ s`
The string.

Returns Returns the length of the string.

Compatibility This function *is* in the C99 specification.
This function is internationally safe to use.
The Palm OS equivalent of this function is the `StrLen()` function; see [Exploring Palm OS: Text and Localization](#). The `StrLen()` function is provided for backward compatibility.

strncat Function

Purpose Concatenates a specified number of characters to a string.

Declared In `posix/string.h`

Prototype `char *strncat (char *s, const char *append, size_t count)`

Parameters `→ s`
The null-terminated string to append to.
`→ append`
The null-terminated string to append.
`→ count`
The number of characters to append.

Returns Returns a pointer to the concatenated string.

Compatibility This function *is* in the C99 specification.

This function is *not* internationally safe to use because it is not multi-byte aware. The result of truncation can be a partial multi-byte character. The Palm OS equivalent of this function is the `StrNCat()` function; see [Exploring Palm OS: Text and Localization](#). However, implementation details of this function differ from the C99 implementation. The `StrNCat()` function does not use the same meaning for the *count* parameter, and thus changing between these two routines requires careful code review.

See Also [strcat\(\)](#)

strncmp Function

Purpose Compares a specified number of characters in two strings.

Declared In `posix/string.h`

Prototype `int strncmp (const char *s1, const char *s2,
size_t len)`

Parameters

- *s1*
The first string to compare.
- *s2*
The second string to compare.
- *len*
The number of characters to compare.

Returns Returns an integer greater than, equal to, or less than zero (0), accordingly as the string *s1* is greater than, equal to, or less than the string *s2*.

Compatibility This function *is* in the C99 specification.

This function is *not* internationally safe to use because it is not multi-byte aware and not locale sensitive. The Palm OS equivalent of this function is the `StrNCompare()` function; see [Exploring Palm OS: Text and Localization](#).

See Also [strcmp\(\)](#)

strncpy Function

Purpose	Copies a specified number of characters in a string.
Declared In	posix/string.h
Prototype	<pre>char *strncpy (char *dst, const char *src, size_t len)</pre>
Parameters	<p>→ <i>dst</i> The destination string.</p> <p>→ <i>src</i> The source string.</p> <p>→ <i>len</i> The number of characters to copy into <i>dst</i>.</p>
Returns	Returns a pointer to the destination string.
Compatibility	This function <i>is</i> in the C99 specification. This function is <i>not</i> internationally safe to use because it is not multi-byte aware. The Palm OS equivalent of this function is the <code>StrNCopy()</code> function; see Exploring Palm OS: Text and Localization .
See Also	strcpy()

strpbrk Function

Purpose	Looks for the first occurrence of any one of an array of characters in a string.
Declared In	posix/string.h
Prototype	<pre>char *strpbrk (const char *s, const char *charset)</pre>
Parameters	<p>→ <i>s</i> The string to check.</p> <p>→ <i>charset</i> The string of characters to search for.</p>
Returns	Returns a pointer to the first occurrence of any character in the string. Otherwise, a NULL pointer is returned if no characters from <i>charset</i> occur anywhere in the string.

Compatibility This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware.

strrchr Function

Purpose Searches a string for the last occurrence of a character.

Declared In `posix/string.h`

Prototype `char *strrchr (const char *s, int c)`

Parameters
→ *b*
The string to search.
→ *c*
The character to search for.

Returns Returns a pointer to the located character, or a NULL pointer if the character does not appear in the string.

Compatibility This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware.

See Also [strchr\(\)](#)

strsep Function

Purpose Locates, in the null-terminated *stringp*, the first occurrence of any character in the string *delim*, and replaces it with a '\0'.

Declared In `posix/string.h`

Prototype `char *strsep (char **stringp, const char *delim)`

Parameters
→ *stringp*
The string to separate.
→ *delim*
The delimiter character.

Returns Returns a pointer to the original value of the string.

Compatibility This function is *not* in the C99 specification.

string.h

strspn

This function *is* a Palm OS extension (not present in C99 or Unix).

This function is *not* internationally safe to use because it is not multi-byte aware.

strspn Function

- Purpose** Spans the initial part of the null-terminated string *s* as long as the characters from *s* occur in string *charset*.
- Declared In** `posix/string.h`
- Prototype** `size_t strspn (const char *s,
const char *charset)`
- Parameters**
- *s*
The string to span.
 - *charset*
The string of characters to search for.
- Returns** Returns the number of characters spanned.
- Compatibility** This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware.
- See Also** [strcspn\(\)](#)

strstr Function

- Purpose** Searches for a string within another.
- Declared In** `posix/string.h`
- Prototype** `char *strstr (const char *big,
const char *little)`
- Parameters**
- *big*
The string to search.
 - *little*
The string to search for within *big*.

- Returns** Returns *big* if *little* is the empty string. Returns a NULL pointer if *little* occurs nowhere in *big*. Otherwise, returns a pointer to the first character of the first occurrence of *little*.
- Compatibility** This function *is* in the C99 specification.
- This function is *not* internationally safe to use because it is not multi-byte aware. The Palm OS equivalent of this function is the `strStr()` function; see [Exploring Palm OS: Text and Localization](#).

strtok Function

- Purpose** Extracts tokens within a string.
- Declared In** `posix/string.h`
- Prototype** `char *strtok (char *str, const char *sep)`
- Parameters**
- *str*
The string to separate.
 - *sep*
The separator string.
- Returns** Returns a pointer to the first token in *str*. Otherwise, a NULL pointer is returned if nothing but separator characters are found.
- Compatibility** This function *is* in the C99 specification.
- This function is *not* internationally safe to use because it is not multi-byte aware.
- See Also** [strtok_r\(\)](#)

strtok_r Function

- Purpose** Re-entrant interface to `strtok()`, which extracts tokens within a string.
- Declared In** `posix/string.h`
- Prototype** `char *strtok_r (char *str, const char *sep, char **lasts)`
- Parameters**
- *str*
The string to separate.

string.h

strxfrm

→ *sep*

The separator string.

→ *lasts*

A user-provided state that needs to be kept between calls to scan the same string.

Returns Returns a pointer to the first token in *str*. Otherwise, a NULL pointer is returned if nothing but separator characters are found.

Compatibility This function is *not* in the C99 specification.

This function is *not* internationally safe to use because it is not multi-byte aware.

See Also [strtok\(\)](#)

strxfrm Function

Purpose Transforms a string into a format that can be passed to `strcmp()` to do locale-sensitive sorting.

Declared In `posix/string.h`

Prototype `size_t strxfrm (char *dst, const char *src, size_t n)`

Parameters → *dst*

The destination string.

→ *src*

The source string.

→ *n*

The number of characters to copy including the null-terminating character.

Returns Returns the length of the transformed string (not including the terminating null character).

Compatibility This function *is* in the C99 specification.

This function is *not* internationally safe to use because it does not yet use Palm OS support for locale-sensitive sorting.

See Also [strcmp\(\)](#)

strings.h

The `<strings.h>` header defines several functions useful for manipulating strings.

Functions and Macros

bcopy Function

Purpose	Performs a byte string copy, coping <i>len</i> bytes from <i>src</i> to <i>dst</i> . The two strings may overlap. If <i>len</i> is zero (0), no bytes are copied.
Declared In	<code>posix/strings.h</code>
Prototype	<code>void bcopy (const void *<i>src</i>, void *<i>dst</i>, size_t <i>len</i>)</code>
Parameters	<code>→ <i>src</i></code> A pointer to the source buffer of bytes. <code>→ <i>dst</i></code> A pointer to the destination buffer of bytes. <code>→ <i>len</i></code> The length of bytes to copy to the specified buffer.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	memcpy()

bzero Function

Purpose	Copies zeroes into the first <i>len</i> bytes of the buffer <i>b</i> .
Declared In	<code>posix/strings.h</code>
Prototype	<code>void bzero (void *<i>b</i>, size_t <i>len</i>)</code>

strings.h

strcasecmp

- Parameters**
- *b*
The buffer that zeroes are copied into.
 - *len*
The length of bytes of zeroes to copy to the specified buffer.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [memset\(\)](#)

strcasecmp Function

- Purpose** Compares the null-terminated strings *s1* and *s2* and returns an integer greater than, equal to, or less than zero (0), accordingly as *s1* is lexicographically greater than, equal to, or less than *s2* after translation of each corresponding character to lowercase.
- Declared In** posix/strings.h
- Prototype** `int strcasecmp (const char *s1, const char *s2)`
- Parameters**
- *s1*
The first string to compare.
 - *s2*
The second string to compare.
- Returns** Returns an integer greater than, equal to, or less than zero (0), accordingly as the string *s1* is greater than, equal to, or less than the string *s2*.
- Compatibility** This function is *not* in the C99 specification.
- This function is *not* internationally safe to use because it is not multi-byte aware and not locale sensitive. The Palm OS equivalents of this function are the `StrNCaselessCompare()` and `TxtCaselessCompare()` functions; see [Exploring Palm OS: Text and Localization](#).
- See Also** [memcmp\(\)](#), [strncasecmp\(\)](#)

strncasecmp Function

- Purpose** Similar to `strcasecmp()`, except compares at most `len` characters.
- Declared In** `posix/strings.h`
- Prototype** `int strncasecmp (const char *s1, const char *s2, size_t len)`
- Parameters**
- `s1`
The first string to compare.
 - `s2`
The second string to compare.
 - `len`
The number of characters to compare.
- Returns** Returns an integer greater than, equal to, or less than zero (0), accordingly as the string `s1` is greater than, equal to, or less than the string `s2`.
- Compatibility** This function is *not* in the C99 specification.
- This function is *not* internationally safe to use because it is not multi-byte aware and not locale sensitive. The Palm OS equivalents of this function are the `StrCaselessCompare()` and `TxtCaselessCompare()` functions; see [Exploring Palm OS: Text and Localization](#).
- See Also** [strcasecmp\(\)](#), [strncmp\(\)](#)

strings.h

strncasecmp

time.h

The `<time.h>` header defines several functions useful for reading and converting the current time and date.

Structures and Types

tm Struct

Purpose	Defines a structure used to hold the time and date.								
Declared In	<code>posix/time.h</code>								
Prototype	<pre>struct tm { int tm_sec; int tm_min; int tm_hour; int tm_mday; int tm_mon; int tm_year; int tm_wday; int tm_yday; int tm_isdst; long tm_gmtoff; char *tm_zone; }</pre>								
Fields	<table><tr><td><code>tm_sec</code></td><td>Seconds after the minute [0-61].</td></tr><tr><td><code>tm_min</code></td><td>Minutes after the hour [0-59].</td></tr><tr><td><code>tm_hour</code></td><td>Hours since midnight [0-23].</td></tr><tr><td><code>tm_mday</code></td><td>Day of the month [1-31].</td></tr></table>	<code>tm_sec</code>	Seconds after the minute [0-61].	<code>tm_min</code>	Minutes after the hour [0-59].	<code>tm_hour</code>	Hours since midnight [0-23].	<code>tm_mday</code>	Day of the month [1-31].
<code>tm_sec</code>	Seconds after the minute [0-61].								
<code>tm_min</code>	Minutes after the hour [0-59].								
<code>tm_hour</code>	Hours since midnight [0-23].								
<code>tm_mday</code>	Day of the month [1-31].								

time.h

Functions and Macros

`tm_mon`
Months since January [0-11].

`tm_year`
Years since 1900.

`tm_wday`
Days since Sunday [0-6].

`tm_yday`
Days since January 1 [0-365].

`tm_isdst`
Daylight Saving Time flag.

`tm_gmtoff`
Offset from UTC in seconds.

`tm_zone`
Timezone abbreviation.

Functions and Macros

asctime Function

Purpose Converts a `tm` structure to a string.

Declared In `posix/time.h`

Prototype `char *asctime (const struct tm *tm)`

Parameters `→ tm`
A `tm` structure.

Returns Returns a pointer to a string that represents the day and time.

Compatibility This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware, not locale sensitive, and contains unlocalized text.

See Also [asctime_r\(\)](#)

asctime_r Function

- Purpose** Re-entrant interface to `asctime()`, which converts a `tm` structure to a string.
- Declared In** `posix/time.h`
- Prototype** `char *asctime_r (const struct tm *tm, char *buf)`
- Parameters**
- *tm*
A `tm` structure.
 - *buf*
The user-provided buffer area, with a size of at least 26 bytes, in which the result is stored.
- Returns** Returns a pointer to a string that represents the day and time.
- Compatibility** This function is *not* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware, not locale sensitive, and contains unlocalized text.
- See Also** [asctime\(\)](#)

clock Function

- Purpose** A program-relative invocation of the system time.
- Declared In** `posix/time.h`
- Prototype** `clock_t clock (void)`
- Returns** Always returns -1, but does not set the global variable `errno`.
- Compatibility** This function *is* in the C99 specification.

ctime Function

- Purpose** Converts a `time_t` type to a string.
- Declared In** `posix/time.h`
- Prototype** `char *ctime (const time_t *clock)`
- Parameters**
- *clock*
The calendar time.

time.h

ctime_r

Returns Returns a pointer to a string representing the local time of the form:

Thu Nov 24 18:22:48 1986\n\0.

Compatibility This function *is* in the C99 specification.

This function is *not* internationally safe to use because it is not multi-byte aware, not locale sensitive, and contains unlocalized text.

See Also [ctime_r\(\)](#)

ctime_r Function

Purpose Re-entrant interface to `ctime()`, which converts a `time_t` type to a string.

Declared In `posix/time.h`

Prototype `char *ctime_r (const time_t *clock, char *buf)`

Parameters \rightarrow *clock*

The calendar time.

\rightarrow *buf*

The user-provided buffer area, with a size of at least 26 bytes, in which the result is stored.

Returns Returns a pointer to a string representing the local time.

Compatibility This function is *not* in the C99 specification.

This function is *not* internationally safe to use because it is not multi-byte aware, not locale sensitive, and contains unlocalized text.

See Also [ctime\(\)](#)

difftime Function

Purpose Computes the difference between two `time_t` types.

Declared In `posix/time.h`

Prototype `double difftime (time_t time1, time_t time0)`

Parameters \rightarrow *time1*

The first time to compare.

→ *time0*

The second time to compare.

Returns Returns the difference between two calendar times, (*time1* - *time0*), expressed in seconds.

Compatibility This function *is* in the C99 specification.

gmtime Function

Purpose Converts a `time_t` value to Coordinated Universal Time (UTC), which is the new name for Greenwich Mean Time.

Declared In `posix/time.h`

Prototype `struct tm *gmtime (const time_t *clock)`

Parameters → *clock*

The calendar time.

Returns Returns a pointer to a `tm` structure upon successful completion. Otherwise, a `NULL` pointer is returned if UTC is not available.

Compatibility This function *is* in the C99 specification.

See Also [gmtime_r\(\)](#)

gmtime_r Function

Purpose Re-entrant interface to `gmtime()`, which converts a `time_t` value to Coordinated Universal Time (UTC), which is the new name for Greenwich Mean Time.

Declared In `posix/time.h`

Prototype `struct tm *gmtime_r (const time_t *clock, struct tm *result)`

Parameters → *clock*

The calendar time.

→ *result*

The user-provided buffer area in which the result is stored.

Returns Returns a pointer to a `tm` structure upon successful completion. Otherwise, a `NULL` pointer is returned if UTC is not available.

Compatibility This function is *not* in the C99 specification.

See Also [gmtime\(\)](#)

localtime Function

Purpose Converts a `time_t` type to a `struct tm` type.

Declared In `posix/time.h`

Prototype `struct tm *localtime (const time_t *clock)`

Parameters \rightarrow `clock`
The calendar time.

Returns Returns a pointer to a `tm` structure representing the local time.

Comments Corrects for the time zone and any time zone adjustments (such as Daylight Saving Time in the U.S.A.).

Compatibility This function *is* in the C99 specification.

See Also [localtime_r\(\)](#), [localtime_tz\(\)](#)

localtime_r Function

Purpose Re-entrant interface to `localtime()`, which converts a `time_t` type to a `struct tm` type.

Declared In `posix/time.h`

Prototype `struct tm *localtime_r (const time_t *clock,
struct tm *result)`

Parameters \rightarrow `clock`
The calendar time.
 \rightarrow `result`
The user-provided buffer area in which the result is stored.

Returns Returns a pointer to a `tm` structure representing the local time.

Comments This function does not imply initialization of the local time conversion information.

Compatibility This function is *not* in the C99 specification.

See Also [localtime\(\)](#)

mktime Function

Purpose	Converts a struct <code>tm</code> type to a <code>time_t</code> type.
Declared In	<code>posix/time.h</code>
Prototype	<code>time_t mktime (struct tm *tm)</code>
Parameters	<p>→ <i>tm</i></p> <p>A <code>tm</code> structure designating a time in the current time zone.</p>
Returns	Returns a string representing a calendar time value.
Compatibility	This function <i>is</i> in the C99 specification.
See Also	<u>mktime_tz()</u>

strftime Function

Purpose	Formats a <code>tm</code> structure to the buffer according to the specified format.
Declared In	<code>posix/time.h</code>
Prototype	<code>size_t strftime (char *buf, size_t maxsize, const char *format, const struct tm *timeptr)</code>
Parameters	<p>→ <i>buf</i></p> <p>The buffer to hold the formatted time.</p> <p>→ <i>maxsize</i></p> <p>The maximum number of characters to be placed into the buffer.</p> <p>→ <i>format</i></p> <p>The format string, consisting of zero or more conversion specifications and ordinary characters.</p> <p>→ <i>timeptr</i></p> <p>A <code>tm</code> structure.</p>
Returns	Returns the number of characters placed into the buffer (not including the terminating null character) if the total number of resulting characters (including the terminating null character) is not more than <i>maxsize</i> . Otherwise, zero (0) is returned and the contents of the array are unknown.
Comments	A conversion specification consists of a “%” character, possibly followed by an E or O modifier, and a terminating conversion

specifier character that determines the conversion specification's behavior. All ordinary characters (including the terminating null byte) are copied unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than *maxsize* bytes are placed into the array. Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined using the LC_TIME category of the current locale and by the values of zero or more members of the broken-down time structure pointed to by *timeptr*, as specified in brackets in the description. If any of the specified values are outside the normal range, the characters stored are unspecified.

The following conversion specifications are supported:

%a

Replaced by the locale's abbreviated weekday name. [*tm_wday*]

%A

Replaced by the locale's full weekday name. [*tm_wday*]

%b

Replaced by the locale's abbreviated month name. [*tm_mon*]

%B

Replaced by the locale's full month name. [*tm_mon*]

%c

Replaced by the locale's appropriate date and time representation.

%C

Replaced by the year divided by 100 and truncated to an integer, as a decimal number [00, 99]. [*tm_year*]

%d

Replaced by the day of the month as a decimal number [01, 31]. [*tm_mday*]

%D

Equivalent to *%m / %d / %y*. [*tm_mon, tm_mday, tm_year*]

%e

Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space. [tm_mday]

%F

Equivalent to %Y - %m - %d (the ISO 8601:2000 standard date format). [tm_year, tm_mon, tm_mday]

%g

Replaced by the last 2 digits of the week-based year as a decimal number [00, 99]. [tm_year, tm_wday, tm_yday]

%G

Replaced by the week-based year as a decimal number (for example, 1977). [tm_year, tm_wday, tm_yday]

%h

Equivalent to %b. [tm_mon]

%H

Replaced by the hour (24-hour clock) as a decimal number [00, 23]. [tm_hour]

%I

Replaced by the hour (12-hour clock) as a decimal number [01, 12]. [tm_hour]

%j

Replaced by the day of the year as a decimal number [001, 366]. [tm_yday]

%m

Replaced by the month as a decimal number [01, 12]. [tm_mon]

%M

Replaced by the minute as a decimal number [00, 59]. [tm_min]

%n

Replaced by a <newline>.

time.h

strftime

%p

Replaced by the locale's equivalent of either a.m. or p.m. [tm_hour]

%r

Replaced by the time in a.m. and p.m. notation. In the POSIX locale, this is equivalent to %I : %M : %S %p. [tm_hour, tm_min, tm_sec]

%R

Replaced by the time in 24-hour notation (%H : %M). [tm_hour, tm_min]

%S

Replaced by the second as a decimal number [00, 60]. [tm_sec]

%t

Replaced by a <tab>.

%T

Replaced by the time (%H : %M : %S). [tm_hour, tm_min, tm_sec]

%u

Replaced by the weekday as a decimal number [1, 7], with 1 representing Monday. [tm_wday]

%U

Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [tm_year, tm_wday, tm_yday]

%V

Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01, 53]. If the week containing January 1 has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [tm_year, tm_wday, tm_yday]

`%w`

Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [`tm_wday`]

`%W`

Replaced by the week number of the year as a decimal number [00, 53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [`tm_year`, `tm_wday`, `tm_yday`]

`%x`

Replaced by the locale's appropriate date representation.

`%X`

Replaced by the locale's appropriate time representation.

`%y`

Replaced by the last two digits of the year as a decimal number [00,99]. [`tm_year`]

`%Y`

Replaced by the year as a decimal number (for example, 1997). [`tm_year`]

`%z`

Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm), or by no characters if no timezone is determinable. For example, "-0430" means 4 hours 30 minutes behind UTC (West of Greenwich). If `tm_isdst` is zero (0), the standard time offset is used. If `tm_isdst` is greater than zero (0), the daylight saving time offset is used. If `tm_isdst` is negative, no characters are returned. [`tm_isdst`]

`%Z`

Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [`tm_isdst`]

`%%`

Replaced by `%`.

If a conversion specification does not correspond to any of the above, the behavior is undefined.

time.h

time

Compatibility This function *is* in the C99 specification.
This function is *not* internationally safe to use because it is not multi-byte aware, not locale sensitive, and contains unlocalized text.

time Function

Purpose Returns the current system calendar time.
Declared In `posix/time.h`
Prototype `time_t time (time_t *timer)`
Parameters \rightarrow *timer*
A `time_t` value.
Returns Returns the current calendar time.
Compatibility This function is *not* in the C99 specification.
See Also [timegm\(\)](#)

timegm Function

Purpose Converts a struct `tm` type to a `time_t` type.
Declared In `posix/time.h`
Prototype `time_t timegm (struct tm *tm)`
Parameters \rightarrow *tm*
A `tm` structure designating a time in Coordinated Universal Time (UTC).
Returns Returns a string representing a calendar time value.
Comments Identical to the `mktime()` function except that while `mktime()` interprets its argument as designating a time in the current time zone, this function interprets its argument as designating a time in Coordinated Universal Time (UTC).
Compatibility This function is *not* in the C99 specification.
See Also [mktime\(\)](#), [time\(\)](#)

time.h

The <time.h> header defines several Palm OS specific functions useful for reading and converting the current time and date.

Constants

TZNAME_MAX

Purpose	Defines the maximum length of a time zone identifier string.
Declared In	posix/sys/time.h
Constants	#define TZNAME_MAX 32

Functions and Macros

getcountrycode Function

Purpose	Gets the two-byte country code for the specified time zone.
Declared In	posix/sys/time.h
Prototype	status_t getcountrycode (const char *tzname, char *buf, size_t bufsize)
Parameters	→ <i>tzname</i> The time zone. → <i>buf</i> The buffer. → <i>bufsize</i> The size of the buffer.
Returns	Returns P_OK upon successful completion; otherwise it returns P_ERROR.

time.h

getgmtoffset

Compatibility This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

getgmtoffset Function

Purpose Gets the difference in seconds between Greenwich Mean Time (GMT) and local standard time.

Declared In `posix/sys/time.h`

Prototype `int32_t getgmtoffset (const char *tzname)`

Parameters `→ tzname`
The time zone.

Returns Returns the current GMT offset, which takes into account daylight saving time. This difference is positive for time zones West of Greenwich and negative for zones East of Greenwich.

Compatibility This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

gettimezone Function

Purpose Copies the current system time zone name into *buf*.

Declared In `posix/sys/time.h`

Prototype `ssize_t gettimezone (char *buf, size_t bufsize)`

Parameters `→ buf`
The buffer.
`→ bufsize`
The size of the buffer.

Returns Returns the number of bytes copied into *buf* upon successful completion; otherwise it returns `P_ERROR`.

Compatibility This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

See Also [hastimezone\(\)](#), [settimezone\(\)](#)

hastimezone Function

- Purpose** Determines if the system has the specified timezone. That is, if a timezone database is installed for the specified timezone.
- Declared In** `posix/sys/time.h`
- Prototype** `int hastimezone (const char *tzname)`
- Parameters** `→ tzname`
The time zone.
- Returns** Returns `P_OK` upon successful completion; otherwise it returns `P_ERROR`.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).
- See Also** [gettimezone\(\)](#), [settimezone\(\)](#)

localtime_tz Function

- Purpose** Converts the specified UTC time in the time zone to a broken-down time.
- Declared In** `posix/sys/time.h`
- Prototype** `void localtime_tz (const time_t *timer,
const char *tzname, struct tm *result)`
- Parameters** `→ timer`
The calendar time.
- `→ tzname`
The time zone.
- `← result`
A `tm` structure.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

mktime_tz Function

- Purpose** Converts a specified broken-down time in the time zone to UTC time. If the `tm_isdst` member of the `tm` struct is negative, this function tries to determine if the specified time zone is currently in daylight saving time.
- Declared In** `posix/sys/time.h`
- Prototype** `time_t mktime_tz (struct tm *tm,
const char *tzname)`
- Parameters**
- *tm*
A `tm` structure.
 - *tzname*
The time zone.
- Returns** Returns the UTC time.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

palm_seconds_to_time_t Function

- Purpose** Takes as input the number of seconds since 1/1/1904 (old Palm epoch) and returns the number of seconds since 1/1/1970 (Unix epoch).
- Declared In** `posix/sys/time.h`
- Prototype** `time_t palm_seconds_to_time_t (uint32_t seconds)`
- Parameters**
- *seconds*
The number of seconds.
- Returns** Returns the number of seconds since 1/1/1970 (Unix epoch).
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).
- See Also** [`time_t_to_palm_seconds\(\)`](#)

settime Function

Purpose	Sets the system time to the specified time.
Declared In	posix/sys/time.h
Prototype	status_t settime (time_t time)
Parameters	→ <i>time</i> The system time.
Returns	Returns P_OK upon successful completion.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).

settimezone Function

Purpose	Sets the system's time zone.
Declared In	posix/sys/time.h
Prototype	status_t settimezone (const char *tzname)
Parameters	→ <i>tzname</i> The time zone.
Returns	Returns P_OK upon successful completion; otherwise it returns P_ERROR.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).
See Also	gettimezone() , hastimezone()

system_real_time Function

Purpose	Gets the value of the real time clock in nanoseconds.
Declared In	posix/sys/time.h
Prototype	nsecs_t system_real_time (void)
Returns	Returns the value of the real time clock in nanoseconds.
Compatibility	This function is <i>not</i> in the C99 specification. This function <i>is</i> a Palm OS extension (not present in C99 or Unix).

system_time Function

- Purpose** Gets the value of the run time clock in nanoseconds.
- Declared In** `posix/sys/time.h`
- Prototype** `nsecs_t system_time (void)`
- Returns** Returns the value of the run time clock in nanoseconds.
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).

time_t_to_palm_seconds Function

- Purpose** Takes as input the number of seconds since 1/1/1970 (Unix epoch) and returns the number of seconds since 1/1/1904 (old Palm epoch).
- Declared In** `posix/sys/time.h`
- Prototype** `uint32_t time_t_to_palm_seconds (time_t seconds)`
- Parameters** `→ seconds`
The number of seconds.
- Returns** Returns the number of seconds since 1/1/1904 (old Palm epoch).
- Compatibility** This function is *not* in the C99 specification.
This function *is* a Palm OS extension (not present in C99 or Unix).
- See Also** [palm_seconds_to_time_t\(\)](#)

uio.h

The <uio.h> header defines two functions useful for vector I/O operations.

Structures and Types

iovec Struct

Purpose	Defines a structure relating to vector I/O information.				
Declared In	posix/sys/uio.h				
Prototype	<pre>struct iovec { void *iov_base; size_t iov_len; }</pre>				
Fields	<table><tr><td><code>iov_base</code></td><td>The base address of a memory region for input or output.</td></tr><tr><td><code>iov_len</code></td><td>The size of the memory pointed to by <code>iov_base</code>.</td></tr></table>	<code>iov_base</code>	The base address of a memory region for input or output.	<code>iov_len</code>	The size of the memory pointed to by <code>iov_base</code> .
<code>iov_base</code>	The base address of a memory region for input or output.				
<code>iov_len</code>	The size of the memory pointed to by <code>iov_base</code> .				

Functions and Macros

readv Function

Purpose	Performs the same action as <code>read()</code> , but scatters the input data into the <code>iovcnt</code> buffers specified by the members of the <code>iov</code> array: <code>iov[0]</code> , <code>iov[1]</code> , ..., <code>iov[iovcnt-1]</code> .
Declared In	posix/sys/uio.h
Prototype	<pre>ssize_t readv (int d, const struct iovec *iov, size_t iovcnt)</pre>

uio.h

writew

- Parameters**
- *d*
The position to start reading from.
 - *iov*
The array.
 - *iovcnt*
The buffer.
- Returns** Returns the number of bytes actually read and placed in the buffer. Zero (0) is returned if end-of-file is read. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [read\(\)](#)

writew Function

- Purpose** Performs the same action as `writew()`, but gathers the output data from the *iovcnt* buffers specified by the members of the *iov* array: `iov[0], iov[1], ..., iov[iovcnt-1]`.
- Declared In** `posix/sys/uio.h`
- Prototype**
`ssize_t writew (int d, const struct iovec *iov, size_t iovcnt)`
- Parameters**
- *d*
The position to start gathering from.
 - *iov*
The array.
 - *iovcnt*
The buffer.
- Returns** Returns the number of bytes actually written. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.
- Compatibility** This function is *not* in the C99 specification.
- See Also** [writew\(\)](#)

unistd.h

The `<unistd.h>` header defines several functions useful for porting code from Unix. These functions are not part of the ANSI C standard.

Functions and Macros

close Function

Purpose	Closes an open file.
Declared In	<code>posix/unistd.h</code>
Prototype	<code>int close (int <i>d</i>)</code>
Parameters	<code>→ <i>d</i></code> The file descriptor.
Returns	Returns zero (0) upon successful completion. Otherwise, -1 is returned and the global variable <code>errno</code> is set to indicate the error.
Compatibility	This function is <i>not</i> in the C99 specification.
See Also	open()

getopt Function

Purpose	Incrementally parses a command line argument list <code>argv</code> and returns the next known option character. An option character is known if it has been specified in the string of accepted option characters, <code>optstring</code> .
Declared In	<code>posix/time.h</code>
Prototype	<code>int getopt (int <i>argc</i>, char * const <i>argv</i>[], const char *<i>optstring</i>)</code>

unistd.h

isatty

- Parameters**
- *argc*
The argument count variable used for command line argument count.
 - *argv*
The command line argument list.
 - *optstring*
The string of accepted option characters.
- Returns** Returns the next known option character.
- Comments** This function does not actually do anything outside of the BCommand environment.
- Compatibility** This function is *not* in the C99 specification.

isatty Function

- Purpose** Determines if a file descriptor refers to a valid terminal type device.
- Declared In** `posix/unistd.h`
- Prototype** `int isatty (int fd)`
- Parameters**
- *fd*
The file descriptor.
- Returns** Returns 1 if *fd* is associated with a terminal device. Otherwise, zero (0) is returned and the global variable `errno` is set to indicate the error.
- Compatibility** This function is *not* in the C99 specification.

read Function

- Purpose** Reads from a file stream that has been opened in binary mode for unformatted input/output.
- Declared In** `posix/unistd.h`
- Prototype** `ssize_t read (int d, void *buf, size_t nbytes)`
- Parameters**
- *d*
The file descriptor.

→ *buf*
The buffer.

→ *nbytes*
The number of bytes of data to read.

Returns Returns zero (0) upon successful completion. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

See Also [write\(\)](#)

write Function

Purpose Writes to a file stream that has been opened in binary mode for unformatted input/output.

Declared In `posix/unistd.h`

Prototype `ssize_t write (int d, const void *buf,
size_t nbytes)`

Parameters → *d*
The file descriptor.

→ *buf*
The buffer.

→ *nbytes*
The number of bytes of data to read.

Returns Returns zero (0) upon successful completion. Otherwise, -1 is returned and the global variable `errno` is set to indicate the error.

Compatibility This function is *not* in the C99 specification.

See Also [read\(\)](#)

unistd.h

write

wchar.h

The `<wchar.h>` header is included for compliance purposes only.

None of the C wide-char (`wchar_t`) functionality is supported in Palm OS. (In fact, the `wchar_t` type is not even used by Palm OS since it can vary in size from 8-bits to 32-bits depending on the compiler.) For safe manipulation of text regardless of the device's character encoding, use the Palm OS String and Text Managers; see [*Exploring Palm OS: Text and Localization*](#).

Index

Symbols

`#define` 15
`#pragma` 17
`__align` 10
`__APGE__` 15
`__APOGEE__` 15
`__arm` 15
`__asm` 11
`__cplusplus` 15
`__DATE__` 16
`__EDG__` 16
`__EDG_VERSION__` 16
`__embedded_cplusplus` 16
`__EXCEPTIONS` 16
`__inline` 11
`__int64` 11
`__pack` 11
`__packed` 11
`__PALMSOURCE__` 16
`__PSI__` 16
`__pure` 11
`__ror32` 11
`__RTTI` 16
`__SIGNED_CHARS__` 16
`__STDC__` 16
`__STDC_HOSTED__` 16
`__STDC_IEC_559__` 5
`__STDC_IEC_559_COMPLEX__` 5
`__STDC_VERSION__` 16
`__TIME__` 17
`__value_in_regs` 11
`__weak` 12
`_BOOL` 15
`_Complex` 5
`_Imaginary` 5
`_PACC_VER` 16
`_WCHAR_T` 17

Numerics

4T architecture 4

A

`abs()` 71, 181

`accept()` 134
`acos()` 71
`acosf()` 72
`acosh()` 72
`acosl()` 72
`addrinfo` 109
`allocate`
 a block of memory heap 187
 space for a group of objects 184
`and` 65
`and_eq` 65
ANSI/ISO/IEC 14882:1998 3
ANSI/ISO/IEC 9899:1999 3
ARM-Thumb Shared Library Architecture 4
`asctime()` 218
`asctime_r()` 219
ASHLA 4
`asin()` 73
`asinf()` 73
`asinh()` 74
`asinl()` 74
`asm` 10
`asprintf()` 149
`assert()` 39
`assert.h` 39
`atan()` 74
`atan2()` 75
`atan2f()` 75
`atan2l()` 76
`atanf()` 76
`atanh()` 77
`atanl()` 77
`atof()` 181
`atoi()` 182
`atol()` 182
`atoll()` 183

B

`bcopy()` 213
`bind()` 135
`bitand` 65
`bitor` 65

break up the floating-point number x into a mantissa and exponent 87, 88

bsearch() 183

bzero() 213

C

c_plusplus 16

C99 3

calloc() 184

cbrt() 77

ceil() 78

ceilf() 78

ceill() 78

change the size of an allocated block of heap memory 190

character set 7

check

- the end-of-file status 151
- the error status 152

classify an argument value as NaN 87

clear a stream's end-of-file 150

clearerr() 150

climits.h 24

clock() 219

close

- a stream 150
- open files 237
- the connection to the database 113
- the TCP connection 112

close() 237

comments 8

compare

- a specified number of characters in two strings 207
- two blocks of memory 197
- two null-terminated strings 214
- two strings 201
- two strings according to locale 201

compl 65

complex.h 24

compute

- an approximation to the sine and cosine of any angle 130
- the absolute value of an integer 181
- the absolute value of x 71, 83, 84
- the arc-cosine of x 71, 72
- the arc-sine of x 73, 74
- the arc-tangent of x 74, 76, 77
- the arc-tangent of y/x 75, 76
- the base-10 logarithm of x 94
- the cosine of x 79, 80, 81
- the cube root of x 77
- the difference between two time_t types 220
- the exponential of x 81, 82
- the floating-point remainder of x/y 85, 86
- the hyperbolic cosine of x 80, 81
- the hyperbolic sine of x 102, 103
- the hyperbolic tangent of x 105, 106
- the inverse hyperbolic cosine of x 72
- the inverse hyperbolic sine of x 74
- the inverse hyperbolic tangent of x 77
- the largest integer not greater than x 84, 85
- the length of a string 206
- the long integer absolute value 186
- the long long integer absolute value 187
- the natural logarithm of x 93, 95, 96
- the nearest 32-bit signed integer not greater than x 129
- the nearest 32-bit signed integer not less than x 129
- the next machine representable number from x in direction y 98
- the non-negative square root of x 103, 104
- the remainder $r := x - n*y$ 100
- the sine of x 101, 102, 103
- the smallest integer not less than x 78
- the $\text{sqrt}(x*x+y*y)$ 89, 90
- the tangent of x 104, 105, 106
- the value of $\exp(x)-1$ 83
- the value of $\log(1+x)$ 95
- x multiplied by 2 to the power n 92, 93
- x raised to power y 98, 99
- $x*(2**n)$ by exponent manipulation 100
- x 's exponent n 90, 95

concatenate

- a specified number of characters to a string 206
- the null-terminated string 204
- two strings 200

connect() 135

constants

- math 127

construct an Internet address 58
conversion specifications 224
convert
 16-bit values between host byte order and network byte order 54
 16-bit values between network byte order and host byte order 55
 32-bit values between host byte order and network byte order 54
 32-bit values between network byte order and host byte order 55
 a character array to a double value 192
 a character array to a long int value 193
 a character array to a long long int value 193
 a character array to an unsigned long long int value 195
 a character array to an unsigned long value 194
 a character string to a double value 181
 a character string to a long long value 183
 a character string to a long value 182
 a character string to an int value 182
 a network format address to presentation format 60
 a presentation format address to network format 61
 a specified broken-down time 232
 a struct tm type to a time_t type 223, 228
 a time_t type to a string 219
 a time_t type to a struct tm type 222
 a time_t value to Coordinated Universal Time (UTC) 221
 a tm structure to a string 218
 lowercase letters 48
 the specified UTC time 231
 uppercase letters 47
copy
 a contiguous memory block 198
 a specified number of characters in a string 208
 characters from the null-terminated string 205
 one string to another 202
 the current system time zone 230
 zeroes into a buffer 213
copysign() 79
cos() 79

cosf() 80
cosh() 80
coshf() 80
coshl() 81
cosl() 81
cpp library 23
ctime() 219
ctime_r() 220
ctype.h 41

D

delete all instances of an environment variable 195
device-specific control functions 63
difftime() 220
div() 184
div_t 179
divide the numerator by the denominator 184, 186
DWARF debugging information 4

E

eabi library 23
endhostent() 112
endnetent() 113
endprotoent() 113
endservent() 113
enter an argument into the environment list 187
erase any input or output buffered 156
errno 49
errno.h 49
ErrTryCatch.h 24
examine the I/O descriptor sets 131
exp() 81
expand an integer constant 147
expf() 82
expl() 82
expm1() 83
extract tokens within a string 211

F

fabs() 83
fabsf() 83
fabsl() 84
fclose() 150

fcntl() 51
fcntl.h 51
fdopen() 151
fenv.h 24
feof() 151
ferror() 152
fflush() 152
fgetc() 152
fgetln() 153
fgetpos() 153
fgets() 154
fileno() 155
find the first sequence of characters in the
 string 202
floor() 84
floorf() 84
floorl() 85
FLT_EVAL 69
flush a stream 152
fmod() 85
fmodf() 86
fmodl() 86
fopen() 155
format a tm structure to the buffer 223
FP_ILOGB0 69
FP_ILOGBNAN 69
FP_INFINITE 69
FP_NAN 69
FP_NORMAL 69
FP_SUBNORMAL 69
FP_ZERO 69
fpclassify() 87
fprintf() 156
fpurge() 156
fputc() 157
fputs() 157
fread() 158
free() 185
freeaddrinfo() 113
freehostent() 114
freopen() 158
frexp() 87
frexpf() 88

frexpl() 88
fscanf() 159
fseek() 160
fseeko() 160
fsetpos() 161
ftell() 161
ftello() 162
fwrite() 162

G

gai_strerror() 114
generate a pseudo-random integer value 189
get
 a character from a stream 152, 163
 a character from stdin 163
 a file position for a stream 153
 a line from a stream 153, 154, 164
 the current value of an environment
 variable 185
 the current value of the file position
 indicator 161
 the difference in seconds 230
 the two-byte country code 229
getaddrinfo() 115
getc() 163
getchar() 163
getcountrycode() 229
getenv() 185
getgmtoffset() 230
gethostbyaddr() 116
gethostbyname() 116
gethostbyname2() 117
gethostent() 117
getipnodebyaddr() 117
getipnodebyname() 118
getnameinfo() 119
getnetbyaddr() 120
getnetbyname() 120
getnetent() 120
getopt() 237
getpeername() 136
getprotobyname() 121
getprotobynumber() 121
getprotoent() 121

gets() 164
getservbyname() 122
getservbyport() 122
getservent() 123
getsockname() 137
getsockopt() 137
gettimezone() 230
getw() 164
global error code variable 49
gmtime() 221
gmtime_r() 221

H

hastimezone() 231
header files
 assert.h 39
 climits.h 24
 complex.h 24
 ctype.h 41
 errno.h 49
 fcntl.h 51
 fenv.h 24
 in.h 53
 inet.h 57
 inttypes.h 24
 ioctl.h 63
 iso646.h 65
 limits.h 24
 locale.h 67
 math.h 69
 namespace.h 24
 netdb.h 109
 PalmMath.h 127
 paths.h 24
 select.h 131
 setjmp.h 24
 signal.h 24
 socket.h 133
 stdarg.h 145
 stddef.h 147
 stdint.h 24
 stdio.h 149
 stdlib.h 179
 string.h 197
 strings.h 213
 termios.h 24

time.h 217, 229
uio.h 235
unistd.h 237
wchar.h 241
hostent 110
hstrerror() 123
htonl() 54
htons() 54
HUGE_VAL 69
HUGE_VALF 69
HUGE_VALL 69
hypot() 89
hypotf() 89
hypotl() 90

I

identifiers 9
ilogb() 90
in.h 53
inet.h 57
inet_addr() 57
inet_aton() 57
inet_lnaof() 58
inet_makeaddr() 58
inet_netof() 59
inet_network() 59
inet_ntoa() 60
inet_ntop() 60
inet_pton() 61
INFINITY 69
inplace_realloc() 185
insert or reset an environment variable 191
interpret the specified character string as an
 Internet address 57
inttypes.h 24
ioctl() 63
ioctl.h 63
iostream 21
iovec 235
isalnum() 41
isalpha() 42
isatty() 238
isblank() 42

iscntrl() 43
isdigit() 43
isfinite() 90
isgraph() 44
isinf() 91
islower() 44
isnan() 91
isnormal() 92
ISO 646 65
iso646.h 65
isprint() 45
ispunct() 45
isspace() 46
isupper() 46
isxdigit() 47

K

keywords 10

L

labs() 186
lceilf() 129
ldexp() 92
ldexpf() 92
ldexpl() 93
ldiv() 186
ldiv_t 179
lfloorf() 129
libc 24, 67
libm.a 70
libraries
 cpp 23
 eabi 23
 pacc 23
 STLport 23
 support 23
limits.h 24
listen() 138
llabs() 187
lldiv_t 180
locale.h 67
localtime() 222
localtime_r() 222

localtime_tz() 231
locate the first occurrence of any character in the
 string 209
log() 93
log10() 94
log10f() 94
log10l() 94
log1p() 95
logb() 95
logf() 95
logl() 96
look for the first occurrence of any one of an array
 of characters 208

M

malloc() 187
manipulate a file descriptor 51
math constants 127
math.h 69
MATH_ERREXCEPT 69
math_errhandling 69
MATH_ERRNO 69
memchr() 197
memcmp() 197
memcpy() 198
memmove() 198
memset() 199
mktime() 223
mktime_tz() 232
modf() 96
modff() 97
modfl() 97

N

namespace.h 24
NAN 69
netdb.h 109
netent 111
nextafter() 98
nodename-to-address translation 115
not 65
not_eq 65
ntohl() 55

ntohs() 55

O

offsetof() 147

open

- a file 51

- and rewind a file 124, 125

open() 51

operators 12

or 65

or_eq 65

output a diagnostic message 39

P

pacc library 23

palm_seconds_to_time_t() 232

PalmMath.h 127

parse a command line argument list 237

paths.h 24

perform

- a binary search 183

- a byte string copy 213

perror() 165

place a character back in a stream 173

pow() 98

powf() 99

powl() 99

preprocessor directives 15

printf() 165

program-relative invocation of the system

- time 219

protoent 111

putc() 166

putchar() 166

putenv() 187

puts() 167

putw() 167

Q

qdiv_t 180

qsort() 188

qsort_r() 188

R

rand() 189

rand_r() 189

random() 190

read

- formatted input from a character string 172

- formatted input from a stream 159

- formatted input from a string 178

- formatted input from stdin 168

- formatted input from stdout 175

- from a file stream 238

- objects from the stream 158

- the next entry in the database 117

- the next line of the file 120, 121, 123

read() 238

readv() 235

realloc() 190

recv() 138

recvfrom() 139

recvmsg() 140

re-entrant interface to

- asctime() 219

- ctime() 220

- gmtime() 221

- localtime() 222

- qsort() 188

- rand() 189

- strerror() 204

- strtok() 211

release

- the dynamically allocated memory 114

- the previously allocated memory to heap 185

remainder() 100

request the use of a connected TCP socket 123

reset the file indicator 168

resize the memory block inplace 185

restrictions

- on C++ 5

- on C99 5

return

- a number suitable for use as an Internet address 57

- a number suitable for use as an Internet network number 59

an ASCII string representing an Internet address 60
 successive pseudo-random numbers 190
 the address of a network host 117
 the current system calendar time 228
 the local network address part (in host order) 58
 the name of a network host 118
 the network number part (in host order) 59
 the socket address structures 113
 x with its sign changed to y's 79
 rewind() 168
 rint() 100
 round x to integral value in floating-point format 100

S

save a copy of a string 203
 scalbn() 100
 scanf() 168
 search

- a string for the last occurrence of a character 209
- for a string within another 210
- for an occurrence of a byte in a buffer 197
- for the first occurrence of a character in a string 200
- for the specified host 116
- until a matching network address is found 120
- until a matching network name is found 120
- until a matching port number is found 122
- until a matching protocol name is found 121, 122
- until a matching protocol number is found 121

 seed

- the random number generator used by rand() 191
- the random number generator used by random() 192

 select() 131
 select.h 131
 send() 141
 sendmsg() 141
 sendto() 142
 separators 15

servent 112
 set

- a file position for a stream 161
- the buffer size and scheme for a stream 170
- the buffer size for a stream 169, 170
- the file position indicator for a stream 160
- the system time 233
- the system's time zone 233

 setbuf() 169
 setbuffer() 169
 setenv() 191
 sethostent() 123
 setjmp.h 24
 setlinebuf() 170
 setnetent() 124
 setprotoent() 124
 setservent() 125
 setsockopt() 143
 settime() 233
 settimezone() 233
 setvbuf() 170
 shutdown() 143
 signal.h 24
 signbit() 101
 sin() 101
 sincosf() 130
 sinf() 102
 sinh() 102
 sinhf() 102
 sinhl() 103
 sinl() 103
 snprintf() 171
 sockaddr 133
 sockaddr_in 53
 socket() 144
 socket.h 133
 socklen_t 133
 sort an array 188
 span the initial part of the null-terminated string 210
 sprintf() 172
 sqrt() 103
 sqrtf() 104
 sqrtl() 104

srand() 191
srandom() 192
sscanf() 172
stdarg.h 145
stddef.h 147
stdint.h 24
stdio.h 149
stdlib.h 179
STLport 21
STLport library 23
strcasemp() 214
strcat() 200
strchr() 200
strcmp() 201
strcoll() 201
strcpy() 202
strcspn() 202
strdup() 203
strerror() 203
strerror_r() 204
strftime() 223
string.h 197
strings.h 213
strlcat() 204
strncpy() 205
strlen() 206
strncasemp() 215
strncat() 206
strncmp() 207
strncpy() 208
strpbrk() 208
strrchr() 209
strsep() 209
strspn() 210
strstr() 210
strtod() 192
strtok() 211
strtok_r() 211
strtol() 193
strtoll() 193
strtoul() 194
strtoull() 195
strxfrm() 212

support library 23
system_real_time() 233
system_time() 234

T

tan() 104
tanf() 105
tanh() 105
tanhf() 106
tanh1() 106
tan1() 106
technical requirements 4
termios.h 24
test
 for a NaN (not a number) 91
 for a negative sign 101
 for a normal value 92
 for alphabetic characters 42
 for alphanumeric characters 41
 for blank-space characters 42
 for control characters 43
 for decimal-digit characters 43
 for finite value 90
 for hexadecimal-digit characters 47
 for infinity (positive or negative) 91
 for lowercase letters 44
 for printing characters 44, 45
 for punctuation characters 45
 for standard white-space characters 46
 for uppercase letters 46
the number of seconds since
 1/1/1904 232
 1/1/1970 234
the value of
 the real time clock 233
 the run time clock 234
time() 228
time.h 217, 229
time_t_to_palm_seconds() 234
timegm() 228
tm 217
tokens 9
tolower() 47
tools documentation viii
toupper() 48

transform a string into a format that can be passed
to strcmp() 212

translate
 address-to-nodename 119
 an error number into an error message 203
TZNAME_MAX 229

U

uio.h 235
ungetc() 173
unistd.h 237
unsetenv() 195

V

va_arg() 145
va_copy() 145
va_end() 146
va_start() 146
vasprintf() 173
vfprintf() 174
vprintf() 175
vscanf() 175
vsnprintf() 176
vsprintf() 177
vsscanf() 178

W

wchar.h 241
wchar_t 241
write
 a character to a stream 166
 a character to an output stream 157
 a character to stdout 166
 a dynamically allocated string 149
 a line to a stream 157
 a string to stdout 167
 an error to stderr 165
 formatted output 156
 formatted output to a character string 171,
 172, 176
 formatted output to a string 177
 formatted output to an output stream 174
 formatted output to stdout 165, 175
 objects to the stream 162
 the specified word to an output stream 167
 to a dynamically allocated string 173
 to a file stream 239
write() 239
writev() 236

X

xor 65
xor_eq 65