## NAME

File::Spec::Mac - File::Spec for Mac OS (Classic)

## SYNOPSIS

```
require File::Spec::Mac; # Done internally by File::Spec if needed
```

## DESCRIPTION

Methods for manipulating file specifications.

## METHODS

canonpath

On Mac OS, there's nothing to be done. Returns what it's given.

*catdir()*

Concatenate two or more directory names to form a path separated by colons (":") ending with a directory. Resulting paths are **relative** by default, but can be forced to be absolute (but avoid this, see below). Automatically puts a trailing ":" on the end of the complete path, because that's what's done in MacPerl's environment and helps to distinguish a file path from a directory path.

**IMPORTANT NOTE:** Beginning with version 1.3 of this module, the resulting path is relative by default and *not* absolute. This decision was made due to portability reasons. Since `File::Spec->catdir()` returns relative paths on all other operating systems, it will now also follow this convention on Mac OS. Note that this may break some existing scripts.

The intended purpose of this routine is to concatenate *directory names*. But because of the nature of Macintosh paths, some additional possibilities are allowed to make using this routine give reasonable results for some common situations. In other words, you are also allowed to concatenate *paths* instead of directory names (strictly speaking, a string like ":a" is a path, but not a name, since it contains a punctuation character ":").

So, beside calls like

```
catdir("a") = ":a:"
catdir("a","b") = ":a:b:"
catdir() = "" (special case)
```

calls like the following

```
catdir(":a:") = ":a:"
catdir(":a","b") = ":a:b:"
catdir(":a:","b") = ":a:b:"
catdir(":a:",":b:") = ":a:b:"
catdir(":") = ":"
```

are allowed.

Here are the rules that are used in `catdir()`; note that we try to be as compatible as possible to Unix:

1. The resulting path is relative by default, i.e. the resulting path will have a leading colon.
2. A trailing colon is added automatically to the resulting path, to denote a directory.
3. Generally, each argument has one leading ":" and one trailing ":" removed (if any). They are then joined together by a ":". Special treatment applies for arguments denoting updir paths like "::lib:", see (4), or arguments consisting solely of colons ("colon paths"), see (5).

4. When an updir path like ‘‘:::lib::’’ is passed as argument, the number of directories to climb up is handled correctly, not removing leading or trailing colons when necessary. E.g.

```
catdir(":::a","::b","c") = ":::a::b:c:"
catdir(":::a::","::b","c") = ":::a:::b:c:"
```

5. Adding a colon ‘‘:’’ or empty string to a path at *any* position doesn't alter the path, i.e. these arguments are ignored. (When a ‘‘ is passed as the first argument, it has a special meaning, see (6)). This way, a colon ’’:‘‘ is handled like a ’’.‘‘ (curdir) on Unix, while an empty string ’’ is generally ignored (see `Unix->canonpath()` ). Likewise, a ‘‘::’’ is handled like a ‘‘..’’ (updir), and a ‘‘:::’’ is handled like a ‘‘../..’’ etc. E.g.

```
catdir("a",":",":","b") = ":a:b:"
catdir("a",":","::",":b") = ":a::b:"
```

6. If the first argument is an empty string or is a volume name, i.e. matches the pattern /^[^:]+:/, the resulting path is **absolute**.

7. Passing an empty string as the first argument to `catdir()` is like passing`File::Spec->rootdir()` as the first argument, i.e.

```
catdir("","a","b") is the same as

catdir(rootdir(),"a","b").
```

This is true on Unix, where `catdir("","a","b")` yields ‘‘/a/b’’ and `rootdir()` is ‘‘/’’. Note that `rootdir()` on Mac OS is the startup volume, which is the closest in concept to Unix’ ‘‘/’’. This should help to run existing scripts originally written for Unix.

8. For absolute paths, some cleanup is done, to ensure that the volume name isn't immediately followed by updirs. This is invalid, because this would go beyond ‘‘root’’. Generally, these cases are handled like their Unix counterparts:

```
Unix:
Unix->catdir("","") = "/"
Unix->catdir("",".") = "/"
Unix->catdir("","..") = "/" # can't go
# beyond root
Unix->catdir("",".","..","..","a") = "/a"
Mac:
Mac->catdir("","") = rootdir() # (e.g. "HD:")
Mac->catdir("",":") = rootdir()
Mac->catdir("","::") = rootdir() # can't go
# beyond root
Mac->catdir("",":","::","::","a") = rootdir() . "a:"
# (e.g. "HD:a:")
```

However, this approach is limited to the first arguments following ‘‘root’’ (again, see `Unix->canonpath()` ). If there are more arguments that move up the directory tree, an invalid path going beyond root can be created.

As you’ve seen, you can force `catdir()` to create an absolute path by passing either an empty string or a path that begins with a volume name as the first argument. However, you are strongly encouraged not to do so, since this is done only for backward compatibility. Newer versions of File::Spec come with a method called `catpath()` (see below), that is designed to

offer a portable solution for the creation of absolute paths. It takes volume, directory and file portions and returns an entire path. While `catdir()` is still suitable for the concatenation of *directory names*, you are encouraged to use `catpath()` to concatenate *volume names* and *directory paths*. E.g.

```
$dir = File::Spec->catdir("tmp","sources");
$abs_path = File::Spec->catpath("MacintoshHD:", $dir,"");
```

yields

```
"MacintoshHD:tmp:sources:" .
```

catfile
Concatenate one or more directory names and a filename to form a complete path ending with a filename. Resulting paths are **relative** by default, but can be forced to be absolute (but avoid this).

**IMPORTANT NOTE:** Beginning with version 1.3 of this module, the resulting path is relative by default and *not* absolute. This decision was made due to portability reasons. Since `File::Spec->catfile()` returns relative paths on all other operating systems, it will now also follow this convention on Mac OS. Note that this may break some existing scripts.

The last argument is always considered to be the file portion. Since `catfile()` uses `catdir()` (see above) for the concatenation of the directory portions (if any), the following with regard to relative and absolute paths is true:

```
catfile("") = ""
catfile("file") = "file"
```

but

```
catfile("","") = rootdir() # (e.g. "HD:")
catfile("","file") = rootdir() . file # (e.g. "HD:file")
catfile("HD:","file") = "HD:file"
```

This means that `catdir()` is called only when there are two or more arguments, as one might expect.

Note that the leading ":" is removed from the filename, so that

```
catfile("a","b","file") = ":a:b:file" and

catfile("a","b",":file") = ":a:b:file"
```

give the same answer.

To concatenate *volume names*, *directory paths* and *filenames*, you are encouraged to use `catpath()` (see below).

curdir
Returns a string representing the current directory. On Mac OS, this is ":".

devnull
Returns a string representing the null device. On Mac OS, this is "Dev:Null".

rootdir
Returns a string representing the root directory. Under MacPerl, returns the name of the startup volume, since that's the closest in concept, although other volumes aren't rooted there. The name has a trailing ":", because that's the correct specification for a volume name on Mac OS.

If Mac::Files could not be loaded, the empty string is returned.

tmpdir
Returns the contents of $ENV{TMPDIR}, if that directory exits or the current working directory otherwise. Under MacPerl, $ENV{TMPDIR} will contain a path like "MacintoshHD:Temporary Items:", which is a hidden directory on your startup volume.

updir
Returns a string representing the parent directory. On Mac OS, this is "::".

file_name_is_absolute
Takes as argument a path and returns true, if it is an absolute path. If the path has a leading ":", it's a relative path. Otherwise, it's an absolute path, unless the path doesn't contain any colons, i.e. it's a name like "a". In this particular case, the path is considered to be relative (i.e. it is considered to be a filename). Use ":" in the appropriate place in the path if you want to distinguish unambiguously. As a special case, the filename '' is always considered to be absolute. Note that with version 1.2 of File::Spec::Mac, this does no longer consult the local filesystem.

E.g.

```
File::Spec->file_name_is_absolute("a"); # false (relative)
File::Spec->file_name_is_absolute(":a:b:"); # false (relative)
File::Spec->file_name_is_absolute("MacintoshHD:");
# true (absolute)
File::Spec->file_name_is_absolute(""); # true (absolute)
```

path
Returns the null list for the MacPerl application, since the concept is usually meaningless under Mac OS. But if you're using the MacPerl tool under MPW, it gives back $ENV{Commands} suitably split, as is done in :lib:ExtUtils:MM_Mac.pm.

splitpath
```
($volume,$directories,$file) = File::Spec->splitpath( $path );
($volume,$directories,$file) = File::Spec->splitpath( $path,
$no_file );
```

Splits a path into volume, directory, and filename portions.

On Mac OS, assumes that the last part of the path is a filename unless $no_file is true or a trailing separator ":" is present.

The volume portion is always returned with a trailing ":". The directory portion is always returned with a leading (to denote a relative path) and a trailing ":" (to denote a directory). The file portion is always returned *without* a leading ":". Empty portions are returned as empty string ''.

The results can be passed to catpath() to get back a path equivalent to (usually identical to) the original path.

splitdir
The opposite of catdir().

```
@dirs = File::Spec->splitdir( $directories );
```

$directories should be only the directory portion of the path on systems that have the concept of a volume or that have path syntax that differentiates files from directories. Consider using splitpath() otherwise.

Unlike just splitting the directories on the separator, empty directory names ("") can be returned. Since catdir() on Mac OS always appends a trailing colon to distinguish a directory path from a file path, a single trailing colon will be ignored, i.e. there's no empty directory name after it.

Hence, on Mac OS, both

```
File::Spec->splitdir( ":a:b::c:" ); and
File::Spec->splitdir( ":a:b::c" );
```

yield:

```
( "a", "b", "::", "c")
```

while

```
File::Spec->splitdir( ":a:b::c::" );
```

yields:

```
( "a", "b", "::", "c", "::")
```

catpath

```
$path = File::Spec->catpath($volume,$directory,$file);
```

Takes volume, directory and file portions and returns an entire path. On Mac OS, `$volume`, `$directory` and `$file` are concatenated. A ':' is inserted if need be. You may pass an empty string for each portion. If all portions are empty, the empty string is returned. If `$volume` is empty, the result will be a relative path, beginning with a ':'. If `$volume` and `$directory` are empty, a leading "":'' (if any) is removed form `$file` and the remainder is returned. If `$file` is empty, the resulting path will have a trailing ':'.

abs2rel

Takes a destination path and an optional base path and returns a relative path from the base path to the destination path:

```
$rel_path = File::Spec->abs2rel( $path ) ;
$rel_path = File::Spec->abs2rel( $path, $base ) ;
```

Note that both paths are assumed to have a notation that distinguishes a directory path (with trailing ':') from a file path (without trailing ':').

If `$base` is not present or '', then the current working directory is used. If `$base` is relative, then it is converted to absolute form using `rel2abs()`. This means that it is taken to be relative to the current working directory.

If `$path` and `$base` appear to be on two different volumes, we will not attempt to resolve the two paths, and we will instead simply return `$path`. Note that previous versions of this module ignored the volume of `$base`, which resulted in garbage results part of the time.

If `$base` doesn't have a trailing colon, the last element of `$base` is assumed to be a filename. This filename is ignored. Otherwise all path components are assumed to be directories.

If `$path` is relative, it is converted to absolute form using `rel2abs()`. This means that it is taken to be relative to the current working directory.

Based on code written by Shigio Yamaguchi.

rel2abs

Converts a relative path to an absolute path:

```
$abs_path = File::Spec->rel2abs( $path ) ;
$abs_path = File::Spec->rel2abs( $path, $base ) ;
```

Note that both paths are assumed to have a notation that distinguishes a directory path (with trailing ':') from a file path (without trailing ':').

If `$base` is not present or '', then `$base` is set to the current working directory. If `$base` is relative, then it is converted to absolute form using `rel2abs()`. This means that it is taken to be relative to the current working directory.

If `$base` doesn't have a trailing colon, the last element of `$base` is assumed to be a filename. This filename is ignored. Otherwise all path components are assumed to be directories.

If `$path` is already absolute, it is returned and `$base` is ignored.

Based on code written by Shigio Yamaguchi.

## AUTHORS

See the authors list in *File::Spec* Mac OS support by Paul Schinder <schinder@pobox.com> and Thomas Wegner <wegner_thomas@yahoo.com>.

## COPYRIGHT



## SEE ALSO

See File::Spec and File::Spec::Unix. This package overrides the implementation of these methods, not the semantics.