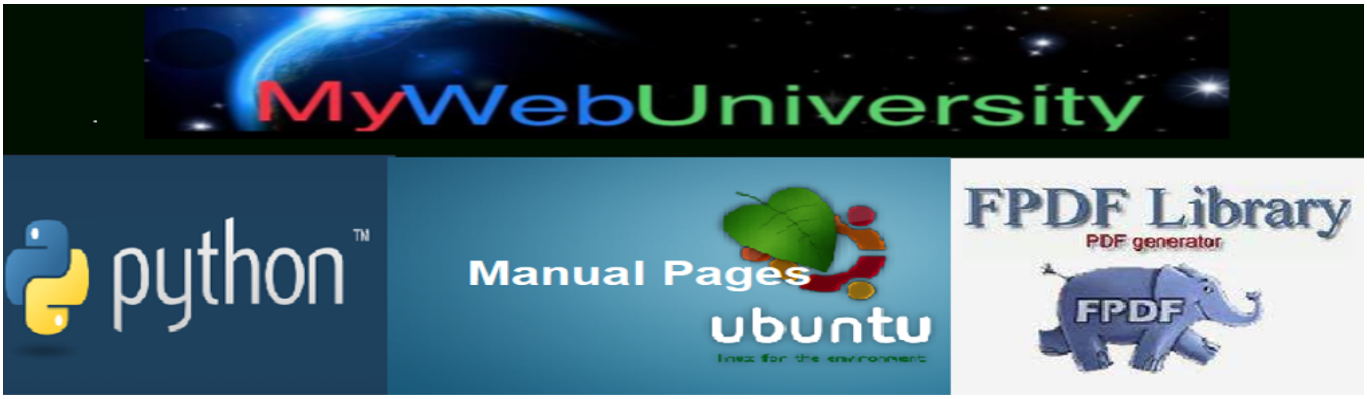

Rocky Enterprise Linux 9.2 Manual Pages on command 'Path::Class.3pm'

$ man Path::Class.3pm

Path::Class(3pm) User Contributed Perl Documentation Path::Class(3pm)

NAME

Path::Class - Cross-platform path specification manipulation

VERSION

version 0.37

SYNOPSIS

```
use Path::Class;
my $dir  = dir('foo', 'bar');       # Path::Class::Dir object
my $file = file('bob', 'file.txt'); # Path::Class::File object
# Stringifies to 'foo/bar' on Unix, 'foo\bar' on Windows, etc.
print "dir: $dir\n";
# Stringifies to 'bob/file.txt' on Unix, 'bob\file.txt' on Windows
print "file: $file\n";
my $subdir  = $dir->subdir('baz');  # foo/bar/baz
my $parent  = $subdir->parent;      # foo/bar
my $parent2 = $parent->parent;      # foo
my $dir2 = $file->dir;              # bob
# Work with foreign paths
use Path::Class qw(foreign_file foreign_dir);
my $file = foreign_file('Mac', ':foo:file.txt');
print $file->dir;                   # :foo:
print $file->as_foreign('Win32');   # foo\file.txt
# Interact with the underlying filesystem:
```

```
  # $dir_handle is an IO::Dir object
  my $dir_handle = $dir->open or die "Can't read $dir: $!";
  # $file_handle is an IO::File object
  my $file_handle = $file->open($mode) or die "Can't read $file: $!";
```

DESCRIPTION

"Path::Class" is a module for manipulation of file and directory specifications (strings describing their locations, like '/home/ken/foo.txt' or 'C:\Windows\Foo.txt') in a cross-platform manner. It supports pretty much every platform Perl runs on, including Unix, Windows, Mac, VMS, Epoc, Cygwin, OS/2, and NetWare.

The well-known module File::Spec also provides this service, but it's sort of awkward to use well, so people sometimes avoid it, or use it in a way that won't actually work properly on platforms significantly different than the ones they've tested their code on.

In fact, "Path::Class" uses "File::Spec" internally, wrapping all the unsightly details so you can concentrate on your application code. Whereas "File::Spec" provides functions for some common path manipulations, "Path::Class" provides an object-oriented model of the world of path specifications and their underlying semantics. "File::Spec" doesn't create any objects, and its classes represent the different ways in which paths must be manipulated on various platforms (not a very intuitive concept). "Path::Class" creates objects representing files and directories, and provides methods that relate them to each other. For instance, the following "File::Spec" code:

```
 my $absolute = File::Spec->file_name_is_absolute(
                  File::Spec->catfile( @dirs, $file )
                );
```

can be written using "Path::Class" as

```
 my $absolute = Path::Class::File->new( @dirs, $file )->is_absolute;
```

or even as

```
 my $absolute = file( @dirs, $file )->is_absolute;
```

Similar readability improvements should happen all over the place when using "Path::Class".

Using "Path::Class" can help solve real problems in your code too - for instance, how many people actually take the "volume" (like "C:" on Windows) into account when writing "File::Spec"-using code? I thought not. But if you use "Path::Class", your file and directory objects will know what volumes they refer to and do the right thing.

The guts of the "Path::Class" code live in the Path::Class::File and Path::Class::Dir modules, so please see those modules' documentation for more details about how to use them.

EXPORT

The following functions are exported by default.

file

A synonym for "Path::Class::File->new".

dir A synonym for "Path::Class::Dir->new".

If you would like to prevent their export, you may explicitly pass an empty list to perl's "use", i.e. "use Path::Class ()".

The following are exported only on demand.

foreign_file

A synonym for "Path::Class::File->new_foreign".

foreign_dir

A synonym for "Path::Class::Dir->new_foreign".

tempdir

Create a new Path::Class::Dir instance pointed to temporary directory.

```
my $temp = Path::Class::tempdir(CLEANUP => 1);
```

A synonym for "Path::Class::Dir->new(File::Temp::tempdir(@_))".

Notes on Cross-Platform Compatibility

Although it is much easier to write cross-platform-friendly code with this module than with "File::Spec", there are still some issues to be aware of.

? On some platforms, notably VMS and some older versions of DOS (I think), all filenames must have an extension. Thus if you create a file called foo/bar and then ask for a list of files in the directory foo, you may find a file called bar. instead of the bar you were expecting. Thus it might be a good idea to use an extension in the first place.

AUTHOR

Ken Williams, KWILLIAMS@cpan.org

COPYRIGHT

SEE ALSO

Path::Class::Dir, Path::Class::File, File::Spec