NAME

Pod::Simple::Search - find POD documents in directory trees

SYNOPSIS

```
use Pod::Simple::Search;
my $name2path = Pod::Simple::Search->new->limit_glob('LWP::*')->survey;
print "Looky see what I found: ",
  join(' ', sort keys %$name2path), "\n";

print "LWPUA docs = ",
  Pod::Simple::Search->new->find('LWP::UserAgent') || "?",
  "\n";
```

DESCRIPTION

Pod::Simple::Search is a class that you use for running searches for Pod files. An object of this class has several attributes (mostly options for controlling search options), and some methods for searching based on those attributes.

The way to use this class is to make a new object of this class, set any options, and then call one of the search options (probably `survey` or `find`). The sections below discuss the syntaxes for doing all that.

CONSTRUCTOR

This class provides the one constructor, called new. It takes no parameters:

```
use Pod::Simple::Search;
my $search = Pod::Simple::Search->new;
```

ACCESSORS

This class defines several methods for setting (and, occasionally, reading) the contents of an object. With two exceptions (discussed at the end of this section), these attributes are just for controlling the way searches are carried out.

Note that each of these return `$self` when you call them as `$self->`*`whatever(value)`*. That's so that you can chain together set-attribute calls like this:

```
my $name2path =
  Pod::Simple::Search->new
  -> inc(0) -> verbose(1) -> callback(\&blab)
  ->survey(@there);
```

...which works exactly as if you'd done this:

```
my $search = Pod::Simple::Search->new;
$search->inc(0);
$search->verbose(1);
$search->callback(\&blab);
my $name2path = $search->survey(@there);
```

$search->inc(*true-or-false*);

This attribute, if set to a true value, means that searches should implicitly add perl's *@INC* paths. This automatically considers paths specified in the `PERL5LIB` environment as this is prepended to *@INC* by the Perl interpreter itself. This attribute's default value is **TRUE**. If you want to search only specific directories, set $self->inc(0) before calling $inc->survey or $inc->find.

$search->verbose(*nonnegative-number*);

This attribute, if set to a nonzero positive value, will make searches output (via `warn`) notes about what they're doing as they do it. This option may be useful for debugging a pod-related module. This attribute's default value is zero, meaning that no `warn` messages are produced. (Setting verbose to 1 turns on some messages, and setting it to 2 turns on even more messages, i.e., makes the following search(es) even more verbose than 1 would make them.)

$search->limit_glob(*some-glob-string*);

This option means that you want to limit the results just to items whose podnames match the given glob/wildcard expression. For example, you might limit your search to just "LWP::*", to search only for modules starting with "LWP::*" (but not including the module "LWP" itself); or you might limit your search to "LW*" to see only modules whose (full) names begin with "LW"; or you might search for "*Find*" to search for all modules with "Find" somewhere in their full name. (You can also use "?" in a glob expression; so "DB?" will match "DBI" and "DBD".)

$search->callback(\&*some_routine*);

This attribute means that every time this search sees a matching Pod file, it should call this callback routine. The routine is called with two parameters: the current file's filespec, and its pod name. (For example: `("/etc/perljunk/File/Crunk.pm", "File::Crunk")` would be in `@_`.)

The callback routine's return value is not used for anything.

This attribute's default value is false, meaning that no callback is called.

$search->laborious(*true-or-false*);

Unless you set this attribute to a true value, Pod::Search will apply Perl-specific heuristics to find the correct module PODs quickly. This attribute's default value is false. You won't normally need to set this to true.

Specifically: Turning on this option will disable the heuristics for seeing only files with Perl-like extensions, omitting subdirectories that are numeric but do *not* match the current Perl interpreter's version ID, suppressing *site_perl* as a module hierarchy name, etc.

$search->shadows(*true-or-false*);

Unless you set this attribute to a true value, Pod::Simple::Search will consider only the first file of a given modulename as it looks thru the specified directories; that is, with this option off, if Pod::Simple::Search has seen a `somepathdir/Foo/Bar.pm` already in this search, then it won't bother looking at a `somelaterpathdir/Foo/Bar.pm` later on in that search, because that file is merely a "shadow". But if you turn on `$self->shadows(1)`, then these "shadow" files are inspected too, and are noted in the pathname2podname return hash.

This attribute's default value is false; and normally you won't need to turn it on.

$search->limit_re(*some-regxp*);

Setting this attribute (to a value that's a regexp) means that you want to limit the results just to items whose podnames match the given regexp. Normally this option is not needed, and the more efficient `limit_glob` attribute is used instead.

$search->dir_prefix(*some-string-value*);

Setting this attribute to a string value means that the searches should begin in the specified subdirectory name (like "Pod" or "File::Find", also expressable as "File/Find"). For example, the search option `$search->limit_glob("File::Find::R*")` is the same as the combination of the search options `$search->limit_re("^File::Find::R") -> dir_prefix("File::Find")`.

Normally you don't need to know about the `dir_prefix` option, but I include it in case it might prove useful for someone somewhere.

(Implementationally, searching with limit_glob ends up setting limit_re and usually dir_prefix.)

$search->progress(*some-progress-object*);

If you set a value for this attribute, the value is expected to be an object (probably of a class that you define) that has a `reach` method and a `done` method. This is meant for reporting progress during the search, if you don't want to use a simple callback.

Normally you don't need to know about the `progress` option, but I include it in case it might prove useful for someone somewhere.

While a search is in progress, the progress object's `reach` and `done` methods are called like this:

```
  # Every time a file is being scanned for pod:
  $progress->reach($count, "Scanning $file");   ++$count;

  # And then at the end of the search:
  $progress->done("Noted $count Pod files total");
```

Internally, we often set this to an object of class Pod::Simple::Progress. That class is probably undocumented, but you may wish to look at its source.

$name2path = $self->name2path;

This attribute is not a search parameter, but is used to report the result of `survey` method, as discussed in the next section.

$path2name = $self->path2name;

This attribute is not a search parameter, but is used to report the result of `survey` method, as discussed in the next section.

MAIN SEARCH METHODS

Once you've actually set any options you want (if any), you can go ahead and use the following methods to search for Pod files in particular ways.

$search->survey(@directories)

The method `survey` searches for POD documents in a given set of files and/or directories. This runs the search according to the various options set by the accessors above. (For example, if the `inc` attribute is on, as it is by default, then the perl @INC directories are implicitly added to the list of directories (if any) that you specify.)

The return value of `survey` is two hashes:

`name2path`

A hash that maps from each pod-name to the filespec (like "Stuff::Thing" => "/whatever/plib/Stuff/Thing.pm")

`path2name`

A hash that maps from each Pod filespec to its pod-name (like "/whatever/plib/Stuff/Thing.pm" => "Stuff::Thing")

Besides saving these hashes as the hashref attributes `name2path` and `path2name`, calling this function also returns these hashrefs. In list context, the return value of `$search->survey` is the list `(\%name2path, \%path2name)`. In scalar context, the return value is `\%name2path`. Or you can just call this in void context.

Regardless of calling context, calling `survey` saves its results in its `name2path` and `path2name` attributes.

E.g., when searching in *$HOME/perl5lib*, the file *$HOME/perl5lib/MyModule.pm* would get the POD name *MyModule*, whereas *$HOME/perl5lib/Myclass/Subclass.pm* would be *Myclass::Subclass*. The name information can be used for POD translators.

Only text files containing at least one valid POD command are found.

In verbose mode, a warning is printed if shadows are found (i.e., more than one POD file with the same POD name is found, e.g. *CPAN.pm* in different directories). This usually indicates duplicate occurrences of modules in the *@INC* search path, which is occasionally inadvertent (but is often simply a case of a user's path dir having a more recent version than the system's general path dirs in general.)

The options to this argument is a list of either directories that are searched recursively, or files. (Usually you wouldn't specify files, but just dirs.) Or you can just specify an empty-list, as in $name2path; with the `inc` option on, as it is by default, teh

The POD names of files are the plain basenames with any Perl-like extension (.pm, .pl, .pod) stripped, and path separators replaced by `::`'s.

Calling Pod::Simple::Search->search(...) is short for Pod::Simple::Search->new->search(...). That is, a throwaway object with default attribute values is used.

$search->simplify_name($str)

The method **simplify_name** is equivalent to **basename**, but also strips Perl-like extensions (.pm, .pl, .pod) and extensions like *.bat*, *.cmd* on Win32 and OS/2, or *.com* on VMS, respectively.

$search->find($pod)
$search->find($pod, @search_dirs)

Returns the location of a Pod file, given a Pod/module/script name (like "Foo::Bar" or "perlvar" or "perldoc"), and an idea of what files/directories to look in. It searches according to the various options set by the accessors above. (For example, if the `inc` attribute is on, as it is by default, then the perl @INC directories are implicitly added to the list of directories (if any) that you specify.)

This returns the full path of the first occurrence to the file. Package names (eg 'A::B') are automatically converted to directory names in the selected directory. Additionally, '.pm', '.pl' and '.pod' are automatically appended to the search as required. (So, for example, under Unix, "A::B" is converted to "somedir/A/B.pm", "somedir/A/B.pod", or "somedir/A/B.pl", as appropriate.)

If no such Pod file is found, this method returns undef.

If any of the given search directories contains a *pod/* subdirectory, then it is searched. (That's how we manage to find *perlfunc*, for example, which is usually in *pod/perlfunc* in most Perl dists.)

The `verbose` and `inc` attributes influence the behavior of this search; notably, `inc`, if true, adds @INC *and also $Config::Config{'scriptdir'}* to the list of directories to search.

It is common to simply say `$filename = Pod::Simple::Search-> new ->find("perlvar")` so that just the @INC (well, and scriptdir) directories are searched. (This happens because the `inc` attribute is true by default.)

Calling Pod::Simple::Search->find(...) is short for Pod::Simple::Search->new->find(...). That is, a throwaway object with default attribute values is used.

$self->contains_pod($file)

Returns true if the supplied filename (not POD module) contains some Pod documentation.

AUTHOR

Sean M. Burke <sburke@cpan.org> borrowed code from Marek Rouchal's Pod::Find, which in turn heavily borrowed code from Nick Ing-Simmons' PodToHtml.

Tim Jenness <t.jenness@jach.hawaii.edu> provided `find` and `contains_pod` to Pod::Find.

SEE ALSO

Pod::Simple, *Pod::Perldoc*