NAME

Module::ScanDeps − Recursively scan Perl code for dependencies

SYNOPSIS

Via the command-line program scandeps:

```
% scandeps *.pm          # Print PREREQ_PM section for *.pm
% scandeps -e "use utf8" # Read script from command line
% scandeps -B *.pm       # Include core modules
% scandeps -V *.pm       # Show autoload/shared/data files
```

Used in a program;

```
use Module::ScanDeps;

# standard usage
my $hash_ref = scan_deps(
    files   => [ 'a.pl', 'b.pl' ],
    recurse => 1,
);

# shorthand; assume recurse == 1
my $hash_ref = scan_deps( 'a.pl', 'b.pl' );

# App::Packer::Frontend compatible interface
# see App::Packer::Frontend for the structure returned by get_files
my $scan = Module::ScanDeps->new;
$scan->set_file( 'a.pl' );
$scan->set_options( add_modules => [ 'Test::More' ] );
$scan->calculate_info;
my $files = $scan->get_files;
```

DESCRIPTION

This module scans potential modules used by perl programs, and returns a hash reference; its keys are the module names as appears in `%INC` (e.g. `Test/More.pm`); the values are hash references with this structure:

```
{
    file    => '/usr/local/lib/perl5/5.8.0/Test/More.pm',
    key     => 'Test/More.pm',
    type    => 'module',    # or 'autoload', 'data', 'shared'
    used_by => [ 'Test/Simple.pm', ... ],
    uses    => [ 'Test/Other.pm', ... ],
}
```

One function, `scan_deps`, is exported by default. Other functions such as (`scan_line`, `scan_chunk`, `add_deps`, `path_to_inc_name`) are exported upon request.

Users of **App::Packer** may also use this module as the dependency-checking frontend, by tweaking their *p2e.pl* like below:

```
use Module::ScanDeps;
...
my $packer = App::Packer->new( frontend => 'Module::ScanDeps' );
...
```

Please see App::Packer::Frontend for detailed explanation on the structure returned by `get_files`.

scan_deps

```
$rv_ref = scan_deps(
    files      => \@files,     recurse => $recurse,
    rv         => \%rv,        skip    => \%skip,
    compile    => $compile,    execute => $execute,
);
$rv_ref = scan_deps(@files); # shorthand, with recurse => 1
```

This function scans each file in `@files`, registering their dependencies into `%rv`, and returns a reference to the updated `%rv`. The meaning of keys and values are explained above.

If `$recurse` is true, `scan_deps` will call itself recursively, to perform a breadth-first search on text files (as defined by the −T operator) found in `%rv`.

If the `\%skip` is specified, files that exists as its keys are skipped. This is used internally to avoid infinite recursion.

If `$compile` or `$execute` is true, runs `files` in either compile-only or normal mode, then inspects their `%INC` after termination to determine additional runtime dependencies.

If `$execute` is an array reference, passes `@$execute` as arguments to each file in `@files` when it is run.

If performance of the scanning process is a concern, `cache_file` can be set to a filename. The scanning results will be cached and written to the file. This will speed up the scanning process on subsequent runs.

Additionally, an option `warn_missing` is recognized. If set to true, `scan_deps` issues a warning to STDERR for every module file that the scanned code depends but that wasn't found. Please note that this may also report numerous false positives. That is why by default, the heuristic silently drops all dependencies it cannot find.

scan_deps_runtime

Like **scan_deps**, but skips the static scanning part.

scan_line

```
@modules = scan_line($line);
```

Splits a line into chunks (currently with the semicolon characters), and return the union of `scan_chunk` calls of them.

If the line is `__END__` or `__DATA__`, a single `__END__` element is returned to signify the end of the program.

Similarly, it returns a single `__POD__` if the line matches `/^=\w/`; the caller is responsible for skipping appropriate number of lines until `=cut`, before calling `scan_line` again.

scan_chunk

```
$module = scan_chunk($chunk);
@modules = scan_chunk($chunk);
```

Apply various heuristics to `$chunk` to find and return the module name(s) it contains. In scalar context, returns only the first module or `undef`.

add_deps

```
$rv_ref = add_deps( rv => \%rv, modules => \@modules );
$rv_ref = add_deps( @modules ); # shorthand, without rv
```

Resolves a list of module names to its actual on-disk location, by finding in `@INC` and `@Module::ScanDeps::IncludeLibs`; modules that cannot be found are skipped.

This function populates the `%rv` hash with module/filename pairs, and returns a reference to it.

path_to_inc_name

```
$perl_name = path_to_inc_name($path, $warn)
```

Assumes `$path` refers to a perl file and does it's best to return the name as it would appear in `%INC`. Returns undef if no match was found and a prints a warning to STDERR if `$warn` is true.

E.g. if `$path` = perl/site/lib/Module/ScanDeps.pm then `$perl_name` will be Module/ScanDeps.pm.

NOTES

`@Module::ScanDeps::IncludeLibs`

You can set this global variable to specify additional directories in which to search modules without modifying `@INC` itself.

`$Module::ScanDeps::ScanFileRE`

You can set this global variable to specify a regular expression to identify what files to scan. By default it includes all files of the following types: .pm, .pl, .t and .al. Additionally, all files without a suffix are considered.

For instance, if you want to scan all files then use the following:

```
$Module::ScanDeps::ScanFileRE = qr/./
```

CAVEATS

This module intentionally ignores the **BSDPAN** hack on FreeBSD — the additional directory is removed from `@INC` altogether.

The static-scanning heuristic is not likely to be 100% accurate, especially on modules that dynamically load other modules.

Chunks that span multiple lines are not handled correctly. For example, this one works:

```
use base 'Foo::Bar';
```

But this one does not:

```
use base
    'Foo::Bar';
```

SEE ALSO

scandeps is a bundled utility that writes `PREREQ_PM` section for a number of files.

An application of **Module::ScanDeps** is to generate executables from scripts that contains prerequisite modules; this module supports two such projects, PAR and App::Packer. Please see their respective documentations on CPAN for further information.

AUTHORS

Audrey Tang <cpan AT audreyt DOT org>

To a lesser degree: Steffen Mueller <smueller AT cpan DOT org>

Parts of heuristics were deduced from:

- **PerlApp** by ActiveState Tools Corp <http://www.activestate.com/>
- **Perl2Exe** by IndigoStar, Inc <http://www.indigostar.com/>

The **scan_deps_runtime** function is contributed by Edward S. Peschko.

You can write to the mailing list at <par AT perl DOT org>, or send an empty mail to <par−subscribe AT perl DOT org> to participate in the discussion.

Please submit bug reports to <bug−Module−ScanDeps AT rt DOT cpan DOT org>.

COPYRIGHT