NAME

Module::Manifest − Parse and examine a Perl distribution MANIFEST file

SYNOPSIS

Open and parse a MANIFEST and MANIFEST.SKIP:

```
my $manifest = Module::Manifest->new( 'MANIFEST', 'MANIFEST.SKIP' );
```

Check if a given file matches any known skip masks:

```
print "yes\n" if $manifest->skipped('.svn');
```

DESCRIPTION

Module::Manifest is a simple utility module created originally for use in Module::Inspector.

It can load a *MANIFEST* file that comes in a Perl distribution tarball, examine the contents, and perform some simple tasks. It can also load the *MANIFEST.SKIP* file and check that.

Granted, the functionality needed to do this is quite simple, but the Perl distribution *MANIFEST* specification contains a couple of little idiosyncracies, such as line comments and space-seperated inline comments.

The use of this module means that any little nigglies are dealt with behind the scenes, and you can concentrate the main task at hand.

Comparison to ExtUtil::Manifest

This module is quite similar to ExtUtils::Manifest, or is at least similar in scope. However, there is a general difference in approach.

ExtUtils::Manifest is imperative, requires the existance of the actual *MANIFEST* file on disk, and requires that your current directory remains the same.

Module::Manifest treats the *MANIFEST* file as an object, can load a the file from anywhere on disk, and can run some of the same functionality without having to change your current directory context.

That said, note that Module::Manifest is aimed at reading and checking existing MANFIFEST files, rather than creating new ones.

COMPATIBILITY

This module should be compatible with Perl 5.005 and above. However, it has only been rigorously tested under Perl 5.10.0 on Linux.

If you encounter any problems on a different version or architecture, please contact the maintainer.

METHODS

new

```
Module::Manifest->new( $manifest, $skip )
```

Creates a `Module::Manifest` object, which either parses the files referenced by the `$manifest` (for MANIFEST) and `$skip` (for MANIFEST.SKIP). If no parameters are specified, it creates an empty object.

Example code:

```
my $manifest = Module::Manifest->new;
my $manifest = Module::Manifest->new( $manifest );
my $manifest = Module::Manifest->new( $manifest, $skip );
```

This method will return an appropriate **Module::Manifest** object or throws an exception on error.

open

```
$manifest->open( $type => $filename )
```

Open and parse the file given by `$filename`, which may be a relative path. The available `$type` options are either: 'skip' or 'manifest'

Example code:

```
$manifest->open( skip => 'MANIFEST.SKIP' );
$manifest->open( manifest => 'MANIFEST' );
```

This method doesn't return anything, but may throw an exception on error.

parse

```
$manifest->parse( $type => \@files )
```

Parse `\@files`, which is an array reference containing a list of files or regular expression masks. The available `$type` options are either: 'skip' or 'manifest'

Example code:

```
$manifest->parse( skip => [
     '\B\.svn\b',
     '^Build$',
     '\bMakefile$',
]);
```

This method doesn't return anything, but may throw an exception on error.

skipped

```
$manifest->skipped( $filename )
```

Check if `$filename` matches any masks that should be skipped, given the regular expressions provided to either the `parse` or `open` methods.

Absolute path names must first be relativized and converted to a Unix-like path string by using the `normalize` method.

Example code:

```
if ($manifest->skipped('Makefile.PL')) {
  # do stuff
}
```

This method returns a boolean true or false value indicating whether the file path is skipped according the `skipfile`.

normalize

```
Module::Manifest->normalize( $path, $rel )
$manifest->normalize( $path, $rel )
```

This method takes a given platform-specific path string and converts it to a Unix-style string compatible with the MANIFEST and MANIFEST.SKIP specifications.

Note that this method normalizes paths depending on the platform detected by `$^O` — that is, Win32 style paths can only be normalized if the module is currently running under Win32.

By default, this method will relativize file paths to the current working directory (using File::Spec's `abs2rel` method without a `$root`). To disable this behaviour, set `$rel` to a false value.

Example code:

```
# Useful for normalizing Win32-style paths
my $normal = Module::Manifest->normalize('t\\test\\file');
# Returns: t/test/file (ie, in Unix style for MANIFEST)
```

This returns a normalized version of the given path.

file

```
$manifest->file
```

The `file` accessor returns the absolute path of the MANIFEST file that was loaded.

skipfile

```
$manifest->skipfile
```

The `skipfile` accessor returns the absolute path of the MANIFEST.SKIP file that was loaded.

dir

```
$manifest->dir
```

The `dir` accessor returns the path to the directory that contains the MANIFEST or skip file, and thus SHOULD be the root of the distribution.

files

```
$manifest->files
```

The `files` method returns the (relative, unix-style) list of files within the manifest. In scalar context, returns the number of files in the manifest.

Example code:

```
my @files = $manifest->files;
```

LIMITATIONS

The directory returned by the `dir` method is overwritten whenever `open` is called. This means that, if MANIFEST and MANIFEST.SKIP are not in the same directory, the module may get a bit confused.

SUPPORT

This module is stored in an Open Repository at the following address:

http://svn.ali.as/cpan/trunk/Module−Manifest <http://svn.ali.as/cpan/trunk/Module-Manifest>

Write access to the repository is made available automatically to any published CPAN author, and to most other volunteers on request.

If you are able to submit your bug report in the form of new (failing) unit tests, or can apply your fix directly instead of submitting a patch, you are **strongly** encouraged to do so. The author currently maintains over 100 modules and it may take some time to deal with non-critical bug reports or patches.

This will guarantee that your issue will be addressed in the next release of the module.

If you cannot provide a direct test or fix, or don't have time to do so, then regular bug reports are still accepted and appreciated via the CPAN bug tracker.

http://rt.cpan.org/NoAuth/ReportBug.html?Queue=Module−Manifest
<http://rt.cpan.org/NoAuth/ReportBug.html?Queue=Module-Manifest>

For other issues, for commercial enhancement and support, or to have your write access enabled for the repository, contact the author at the email address above.

AUTHOR

Adam Kennedy <adamk AT cpan DOT org>

CONTIRBUTORS

Jonathan Yu <jawnsy AT cpan DOT org>

SEE ALSO

ExtUtils::Manifest

COPYRIGHT