NAME

Parse::CPAN::Meta - Parse META.yml and META.json CPAN metadata files

VERSION

version 2.150010

SYNOPSIS

```
######################################################
# In your file

---
name: My-Distribution
version: 1.23
resources:
  homepage: "http://example.com/dist/My-Distribution"

######################################################
# In your program

use Parse::CPAN::Meta;

my $distmeta = Parse::CPAN::Meta->load_file('META.yml');

# Reading properties
my $name     = $distmeta->{name};
my $version  = $distmeta->{version};
my $homepage = $distmeta->{resources}{homepage};
```

DESCRIPTION

Parse::CPAN::Meta is a parser for *META.json* and *META.yml* files, using *JSON::PP* and/or *CPAN::Meta::YAML*.

Parse::CPAN::Meta provides three methods: `load_file`, `load_json_string`, and `load_yaml_string`. These will read and deserialize CPAN metafiles, and are described below in detail.

Parse::CPAN::Meta provides a legacy API of only two functions, based on the YAML functions of the same name. Wherever possible, identical calling semantics are used. These may only be used with YAML sources.

All error reporting is done with exceptions (die'ing).

Note that META files are expected to be in UTF-8 encoding, only. When converted string data, it must first be decoded from UTF-8.

METHODS

load_file

```
my $metadata_structure = Parse::CPAN::Meta->load_file('META.json');

my $metadata_structure = Parse::CPAN::Meta->load_file('META.yml');
```

This method will read the named file and deserialize it to a data structure, determining whether it should be JSON or YAML based on the filename. The file will be read using the ":utf8" IO layer.

load_yaml_string

```
  my $metadata_structure =
Parse::CPAN::Meta->load_yaml_string($yaml_string);
```

This method deserializes the given string of YAML and returns the first document in it. (CPAN metadata files should always have only one document.) If the source was UTF-8 encoded, the string must be decoded before calling `load_yaml_string`.

load_json_string

```
  my $metadata_structure =
Parse::CPAN::Meta->load_json_string($json_string);
```

This method deserializes the given string of JSON and the result. If the source was UTF-8 encoded, the string must be decoded before calling `load_json_string`.

load_string

```
  my $metadata_structure = Parse::CPAN::Meta->load_string($some_string);
```

If you don't know whether a string contains YAML or JSON data, this method will use some heuristics and guess. If it can't tell, it assumes YAML.

yaml_backend

```
  my $backend = Parse::CPAN::Meta->yaml_backend;
```

Returns the module name of the YAML serializer. See *ENVIRONMENT* for details.

json_backend

```
  my $backend = Parse::CPAN::Meta->json_backend;
```

Returns the module name of the JSON serializer. If `CPAN_META_JSON_BACKEND` is set, this will be whatever that's set to. If not, this will either be *JSON::PP* or *JSON*. If `PERL_JSON_BACKEND` is set, this will return *JSON* as further delegation is handled by the *JSON* module. See *ENVIRONMENT* for details.

json_decoder

```
  my $decoder = Parse::CPAN::Meta->json_decoder;
```

Returns the module name of the JSON decoder. Unlike *json_backend*, this is not necessarily a full *JSON*-style module, but only something that will provide a `decode_json` subroutine. If `CPAN_META_JSON_DECODER` is set, this will be whatever that's set to. If not, this will be whatever has been selected as *json_backend*. See *ENVIRONMENT* for more notes.

FUNCTIONS

For maintenance clarity, no functions are exported by default. These functions are available for backwards compatibility only and are best avoided in favor of `load_file`.

Load

```
  my @yaml = Parse::CPAN::Meta::Load( $string );
```

Parses a string containing a valid YAML stream into a list of Perl data structures.

LoadFile

```
my @yaml = Parse::CPAN::Meta::LoadFile( 'META.yml' );
```

Reads the YAML stream from a file instead of a string.

ENVIRONMENT

CPAN_META_JSON_DECODER

By default, *JSON::PP* will be used for deserializing JSON data. If the `CPAN_META_JSON_DECODER` environment variable exists, this is expected to be the name of a loadable module that provides a `decode_json` subroutine, which will then be used for deserialization. Relying only on the existence of said subroutine allows for maximum compatibility, since this API is provided by all of *JSON::PP*, *JSON::XS*, *Cpanel::JSON::XS*, *JSON::MaybeXS*, *JSON::Tiny*, and *Mojo::JSON*.

CPAN_META_JSON_BACKEND

By default, *JSON::PP* will be used for deserializing JSON data. If the `CPAN_META_JSON_BACKEND` environment variable exists, this is expected to be the name of a loadable module that provides the *JSON* API, since downstream code expects to be able to call `new` on this class. As such, while *JSON::PP*, *JSON::XS*, *Cpanel::JSON::XS* and *JSON::MaybeXS* will work for this, to use *Mojo::JSON* or *JSON::Tiny* for decoding requires setting *CPAN_META_JSON_DECODER*.

PERL_JSON_BACKEND

If the `CPAN_META_JSON_BACKEND` environment variable does not exist, and if `PERL_JSON_BACKEND` environment variable exists, is true and is not "JSON::PP", then the *JSON* module (version 2.5 or greater) will be loaded and used to interpret `PERL_JSON_BACKEND`. If *JSON* is not installed or is too old, an exception will be thrown. Note that at the time of writing, the only useful values are 1, which will tell *JSON* to guess, or *JSON::XS* - if you want to use a newer JSON module, see *CPAN_META_JSON_BACKEND*.

PERL_YAML_BACKEND

By default, *CPAN::Meta::YAML* will be used for deserializing YAML data. If the `PERL_YAML_BACKEND` environment variable is defined, then it is interpreted as a module to use for deserialization. The given module must be installed, must load correctly and must implement the `Load()` function or an exception will be thrown.

AUTHORS

- David Golden <dagolden@cpan.org>
- Ricardo Signes <rjbs@cpan.org>
- Adam Kennedy <adamk@cpan.org>

COPYRIGHT AND LICENSE