NAME

CPAN::Meta - the distribution metadata for a CPAN dist

VERSION

version 2.140640

SYNOPSIS

```
use v5.10;
use strict;
use warnings;
use CPAN::Meta;
use Module::Load;

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

printf "testing requirements for %s version %s\n",
$meta->name,
$meta->version;

my $prereqs = $meta->effective_prereqs;

for my $phase ( qw/configure runtime build test/ ) {
say "Requirements for $phase:";
my $reqs = $prereqs->requirements_for($phase, "requires");
for my $module ( sort $reqs->required_modules ) {
my $status;
if ( eval { load $module unless $module eq 'perl'; 1 } ) {
my $version = $module eq 'perl' ? $] : $module->VERSION;
$status = $reqs->accepts_module($module, $version)
? "$version ok" : "$version not ok";
} else {
$status = "missing"
};
say " $module ($status)";
}
}
```

DESCRIPTION

Software distributions released to the CPAN include a *META.json* or, for older distributions, *META.yml*, which describes the distribution, its contents, and the requirements for building and installing the distribution. The data structure stored in the *META.json* file is described in CPAN::Meta::Spec.

CPAN::Meta provides a simple class to represent this distribution metadata (or *distmeta*), along with some helpful methods for interrogating that data.

The documentation below is only for the methods of the CPAN::Meta object. For information on the meaning of individual fields, consult the spec.

METHODS

new

```
my $meta = CPAN::Meta->new($distmeta_struct, \%options);
```

Returns a valid CPAN::Meta object or dies if the supplied metadata hash reference fails to validate. Older-format metadata will be up-converted to version 2 if they validate against the original stated specification.

It takes an optional hashref of options. Valid options include:

- lazy_validation — if true, new will attempt to convert the given metadata to version 2 before attempting to validate it. This means than any fixable errors will be handled by CPAN::Meta::Converter before validation. (Note that this might result in invalid optional data being silently dropped.) The default is false.

create

```
my $meta = CPAN::Meta->create($distmeta_struct, \%options);
```

This is same as `new()`, except that `generated_by` and `meta-spec` fields will be generated if not provided. This means the metadata structure is assumed to otherwise follow the latest CPAN::Meta::Spec.

load_file

```
my $meta = CPAN::Meta->load_file($distmeta_file, \%options);
```

Given a pathname to a file containing metadata, this deserializes the file according to its file suffix and constructs a new `CPAN::Meta` object, just like `new()`. It will die if the deserialized version fails to validate against its stated specification version.

It takes the same options as `new()` but `lazy_validation` defaults to true.

load_yaml_string

```
my $meta = CPAN::Meta->load_yaml_string($yaml, \%options);
```

This method returns a new CPAN::Meta object using the first document in the given YAML string. In other respects it is identical to `load_file()`.

load_json_string

```
my $meta = CPAN::Meta->load_json_string($json, \%options);
```

This method returns a new CPAN::Meta object using the structure represented by the given JSON string. In other respects it is identical to `load_file()`.

load_string

```
my $meta = CPAN::Meta->load_string($string, \%options);
```

If you don't know if a string contains YAML or JSON, this method will use Parse::CPAN::Meta to guess. In other respects it is identical to `load_file()`.

save

```
$meta->save($distmeta_file, \%options);
```

Serializes the object as JSON and writes it to the given file. The only valid option is `version`, which defaults to '2'. On Perl 5.8.1 or later, the file is saved with UTF-8 encoding.

For `version` 2 (or higher), the filename should end in '.json'. JSON::PP is the default JSON backend. Using another JSON backend requires JSON 2.5 or later and you must set the `$ENV{PERL_JSON_BACKEND}` to a supported alternate backend like JSON::XS

For `version` less than 2, the filename should end in '.yml'. CPAN::Meta::Converter is used to generate an older metadata structure, which is serialized to YAML. CPAN::Meta::YAML is the default YAML backend. You may set the `$ENV{PERL_YAML_BACKEND}` to a supported alternative backend, though this is not recommended due to subtle incompatibilities between YAML parsers on CPAN.

meta_spec_version

This method returns the version part of the `meta_spec` entry in the distmeta structure. It is equivalent to:

```
$meta->meta_spec->{version};
```

effective_prereqs

```
my $prereqs = $meta->effective_prereqs;

my $prereqs = $meta->effective_prereqs( \@feature_identifiers );
```

This method returns a CPAN::Meta::Prereqs object describing all the prereqs for the distribution. If an arrayref of feature identifiers is given, the prereqs for the identified features are merged together with the distribution's core prereqs before the CPAN::Meta::Prereqs object is returned.

should_index_file

```
... if $meta->should_index_file( $filename );
```

This method returns true if the given file should be indexed. It decides this by checking the `file` and `directory` keys in the `no_index` property of the distmeta structure.

`$filename` should be given in unix format.

should_index_package

```
... if $meta->should_index_package( $package );
```

This method returns true if the given package should be indexed. It decides this by checking the `package` and `namespace` keys in the `no_index` property of the distmeta structure.

features

```
my @feature_objects = $meta->features;
```

This method returns a list of CPAN::Meta::Feature objects, one for each optional feature described by the distribution's metadata.

feature

```
my $feature_object = $meta->feature( $identifier );
```

This method returns a CPAN::Meta::Feature object for the optional feature with the given identifier. If no feature with that identifier exists, an exception will be raised.

as_struct

```
my $copy = $meta->as_struct( \%options );
```

This method returns a deep copy of the object's metadata as an unblessed hash reference. It takes an optional hashref of options. If the hashref contains a `version` argument, the copied metadata will be converted to the version of the specification and returned. For example:

```
my $old_spec = $meta->as_struct( {version => "1.4"} );
```

as_string

```
my $string = $meta->as_string( \%options );
```

This method returns a serialized copy of the object's metadata as a character string. (The strings are **not** UTF-8 encoded.) It takes an optional hashref of options. If the hashref contains a `version` argument, the copied metadata will be converted to the version of the specification and returned. For example:

```
my $string = $meta->as_string( {version => "1.4"} );
```

For `version` greater than or equal to 2, the string will be serialized as JSON. For `version` less than 2, the string will be serialized as YAML. In both cases, the same rules are followed as in the `save()` method for choosing a serialization backend.

STRING DATA

The following methods return a single value, which is the value for the corresponding entry in the distmeta structure. Values should be either undef or strings.

- abstract
- description
- dynamic_config
- generated_by
- name

- release_status
- version

LIST DATA

These methods return lists of string values, which might be represented in the distmeta structure as arrayrefs or scalars:

- authors
- keywords
- licenses

The `authors` and `licenses` methods may also be called as `author` and `license`, respectively, to match the field name in the distmeta structure.

MAP DATA

These readers return hashrefs of arbitrary unblessed data structures, each described more fully in the specification:

- meta_spec
- resources
- provides
- no_index
- prereqs
- optional_features

CUSTOM DATA

A list of custom keys are available from the `custom_keys` method and particular keys may be retrieved with the `custom` method.

```
say $meta->custom($_) for $meta->custom_keys;
```

If a custom key refers to a data structure, a deep clone is returned.

BUGS

Please report any bugs or feature using the CPAN Request Tracker. Bugs can be submitted through the web interface at <http://rt.cpan.org/Dist/Display.html?Queue=CPAN-Meta>

When submitting a bug or request, please include a test-file or a patch to an existing test-file that illustrates the bug or desired feature.

SEE ALSO

- CPAN::Meta::Converter
- CPAN::Meta::Validator

SUPPORT

Bugs / Feature Requests

Please report any bugs or feature requests through the issue tracker at <https://github.com/Perl-Toolchain-Gang/CPAN-Meta/issues>. You will be notified automatically of any progress on your issue.

Source Code

This is open source software. The code repository is available for public review and contribution under the terms of the license.

<https://github.com/Perl-Toolchain-Gang/CPAN-Meta>

```
git clone https://github.com/Perl-Toolchain-Gang/CPAN-Meta.git
```

AUTHORS

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

CONTRIBUTORS

- Ansgar Burchardt <ansgar@cpan.org>
- Avar Arnfjord Bjarmason <avar@cpan.org>
- Christopher J. Madsen <cjm@cpan.org>
- Chuck Adams <cja987@gmail.com>
- Cory G Watson <gphat@cpan.org>
- Damyan Ivanov <dam@cpan.org>
- Eric Wilhelm <ewilhelm@cpan.org>
- Gregor Hermann <gregoa@debian.org>
- Karen Etheridge <ether@cpan.org>
- Ken Williams <kwilliams@cpan.org>
- Kenichi Ishigaki <ishigaki@cpan.org>
- Lars Dieckow <daxim@cpan.org>
- Leon Timmermans <leont@cpan.org>
- Mark Fowler <markf@cpan.org>
- Michael G. Schwern <mschwern@cpan.org>
- Olaf Alders <olaf@wundersolutions.com>
- Olivier Mengue <dolmen@cpan.org>
- Randy Sims <randys@thepierianspring.org>

COPYRIGHT AND LICENSE