NAME

YAML::XS – Perl YAML Serialization using XS and libyaml

SYNOPSIS

```
use YAML::XS;

my $yaml = Dump [ 1..4 ];
my $array = Load $yaml;
```

DESCRIPTION

Kirill Simonov's `libyaml` is arguably the best YAML implementation. The C library is written precisely to the YAML 1.1 specification. It was originally bound to Python and was later bound to Ruby.

This module is a Perl XS binding to libyaml which offers Perl the best YAML support to date.

This module exports the functions `Dump`, `Load`, `DumpFile` and `LoadFile`. These functions are intended to work exactly like `YAML.pm`'s corresponding functions. Only `Load` and `Dump` are exported by default.

CONFIGURATION

`$YAML::XS::LoadBlessed` (since v0.69)

Default: false.

The default was changed in version 0.81.

When set to false, it will not bless data into objects, which can be a security problem, when loading YAML from an untrusted source. It will silently ignore the tag and just load the data unblessed.

In PyYAML, this is called SafeLoad.

If set to true, it will load the following YAML as objects:

```
---
local: !Foo::Bar [a]
perl: !!perl/hash:Foo::Bar { a: 1 }
regex: !!perl/regexp:Foo::Bar pattern
```

You can create any kind of object with YAML. The creation itself is not the critical part. If the class has a `DESTROY` method, it will be called once the object is deleted. An example with File::Temp removing files can be found at <https://bugs.debian.org/cgi−bin/bugreport.cgi?bug=862373>

`$YAML::XS::UseCode`
`$YAML::XS::DumpCode`
`$YAML::XS::LoadCode`

If enabled supports deparsing and evaling of code blocks.

Note that support for loading code was added in version 0.75, although `$LoadCode` was documented already in earlier versions.

`$YAML::XS::QuoteNumericStrings`

When true (the default) strings that look like numbers but have not been numified will be quoted when dumping.

This ensures leading that things like leading zeros and other formatting are preserved.

`$YAML::XS::Boolean` (since v0.67)

Default is undef.

When set to `"JSON::PP"` or `"boolean"`, the plain (unquoted) strings `true` and `false` will be loaded as `JSON::PP::Boolean` or `boolean.pm` objects. Those objects will be dumped again as plain ‘‘true’’ or ‘‘false’’.

It will try to load JSON::PP or boolean and die if it can't be loaded.

With that it's possible to add new ‘‘real’’ booleans to a data structure:

```
local $YAML::XS::Boolean = "JSON::PP"; # or "boolean"
my $data = Load("booltrue: true");
$data->{boolfalse} = JSON::PP::false;
my $yaml = Dump($data);
# boolfalse: false
# booltrue: true
```

It also lets booleans survive when loading YAML via YAML::XS and encode it in JSON via one of the various JSON encoders, which mostly support JSON::PP booleans.

Please note that JSON::PP::Boolean and boolean.pm behave a bit differently. Ideally you should only use them in boolean context.

If not set, booleans are loaded as special perl variables `PL_sv_yes` and `PL_sv_no`, which have the disadvantage that they are readonly, and you can't add those to an existing data structure with pure perl.

If you simply need to load ''perl booleans'' that are true or false in boolean context, you will be fine with the default setting.

`$YAML::XS::Indent` (since v0.76)
Default is 2.

Sets the number of spaces for indentation for `Dump`.

USING YAML::XS WITH UNICODE

Handling unicode properly in Perl can be a pain. YAML::XS only deals with streams of utf8 octets. Just remember this:

```
$perl = Load($utf8_octets);
$utf8_octets = Dump($perl);
```

There are many, many places where things can go wrong with unicode. If you are having problems, use Devel::Peek on all the possible data points.

LIBYAML

You can find out (since v.079) which libyaml version this module was built with:

```
my $libyaml_version = YAML::XS::LibYAML::libyaml_version();
```

SEE ALSO

- YAML.pm
- YAML::Syck
- YAML::Tiny
- YAML::PP
- YAML::PP::LibYAML

AUTHOR

Ingy döt Net <ingy@cpan.org>

COPYRIGHT AND LICENSE