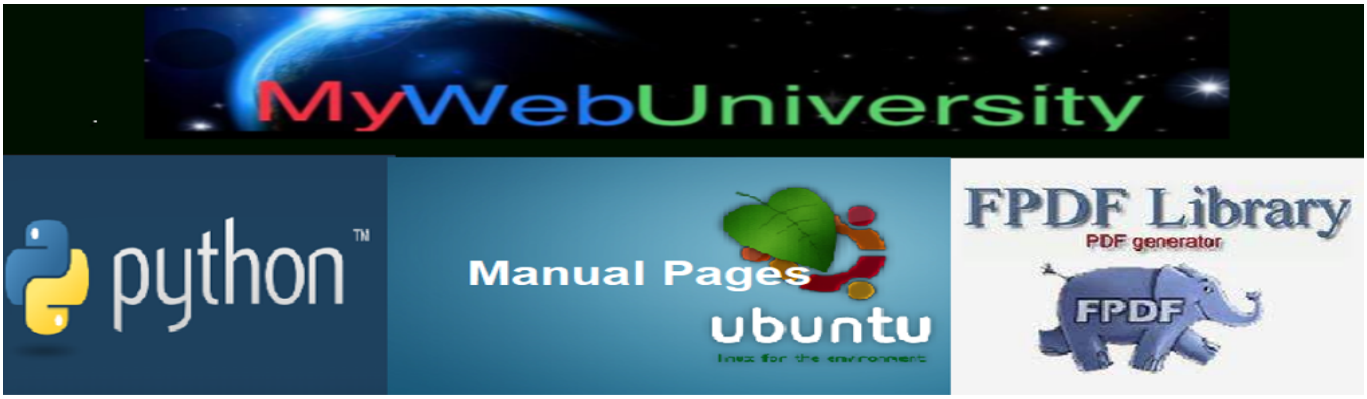

Rocky Enterprise Linux 9.2 Manual Pages on command 'YAML.3pm'

$ man YAML.3pm

YAML(3pm) User Contributed Perl Documentation YAML(3pm)

NAME

YAML - YAML Ain't Markup Language?

VERSION

This document describes YAML version 1.30.

NOTE

This module has been released to CPAN as YAML::Old, and soon YAML.pm will be changed to just be a frontend interface module for all the various Perl YAML implementation modules, including YAML::Old.

If you want robust and fast YAML processing using the normal Dump/Load API, please consider switching to YAML::XS. It is by far the best Perl module for YAML at this time.

It requires that you have a C compiler, since it is written in C.

If you really need to use this version of YAML.pm it will always be available as YAML::Old.

The rest of this documentation is left unchanged, until YAML.pm is switched over to the new UI-only version.

SYNOPSIS

```
use YAML;
# Load a YAML stream of 3 YAML documents into Perl data structures.
my ($hashref, $arrayref, $string) = Load(<<'...');
---
name: ingy       # A Mapping
age: old
```

```
    weight: heavy
    # I should comment that I also like pink, but don't tell anybody.
    favorite colors:
      - red
      - green
      - blue
    ---
    - Clark Evans    # A Sequence
    - Oren Ben-Kiki
    - Ingy d?t Net
    --- >            # A Block Scalar
    You probably think YAML stands for "Yet Another Markup Language". It
    ain't! YAML is really a data serialization language. But if you want
    to think of it as a markup, that's OK with me. A lot of people try
    to use XML as a serialization format.
    "YAML" is catchy and fun to say. Try it. "YAML, YAML, YAML!!!"
    ...
    # Dump the Perl data structures back into YAML.
    print Dump($string, $arrayref, $hashref);
    # YAML::Dump is used the same way you'd use Data::Dumper::Dumper
    use Data::Dumper;
    print Dumper($string, $arrayref, $hashref);
    Since version 1.25 YAML.pm supports trailing comments.
```

DESCRIPTION

The YAML.pm module implements a YAML Loader and Dumper based on the YAML 1.0 specification. <http://www.yaml.org/spec/>

YAML is a generic data serialization language that is optimized for human readability. It can be used to express the data structures of most modern programming languages. (Including Perl!!!)

For information on the YAML syntax, please refer to the YAML specification.

WHY YAML IS COOL

YAML is readable for people.

It makes clear sense out of complex data structures. You should find that YAML is an

exceptional data dumping tool. Structure is shown through indentation, YAML supports recursive data, and hash keys are sorted by default. In addition, YAML supports several styles of scalar formatting for different types of data.

YAML is editable.

YAML was designed from the ground up to be an excellent syntax for configuration files. Almost all programs need configuration files, so why invent a new syntax for each one? And why subject users to the complexities of XML or native Perl code?

YAML is multilingual.

Yes, YAML supports Unicode. But I'm actually referring to programming languages. YAML was designed to meet the serialization needs of Perl, Python, Ruby, Tcl, PHP, Javascript and Java. It was also designed to be interoperable between those languages. That means YAML serializations produced by Perl can be processed by Python.

YAML is taint safe.

Using modules like Data::Dumper for serialization is fine as long as you can be sure that nobody can tamper with your data files or transmissions. That's because you need to use Perl's "eval()" built-in to deserialize the data. Somebody could add a snippet of Perl to erase your files.

YAML's parser does not need to eval anything.

YAML is full featured.

YAML can accurately serialize all of the common Perl data structures and deserialize them again without losing data relationships. Although it is not 100% perfect (no serializer is or can be perfect), it fares as well as the popular current modules: Data::Dumper, Storable, XML::Dumper and Data::Denter.

YAML.pm also has the ability to handle code (subroutine) references and typeglobs. (Still experimental) These features are not found in Perl's other serialization modules.

YAML is extensible.

The YAML language has been designed to be flexible enough to solve it's own problems. The markup itself has 3 basic construct which resemble Perl's hash, array and scalar. By default, these map to their Perl equivalents. But each YAML node also supports a tagging mechanism (type system) which can cause that node to be interpreted in a completely different manner. That's how YAML can support object serialization and oddball structures like Perl's typeglob.

YAML IMPLEMENTATIONS IN PERL

This module, YAML.pm, is really just the interface module for YAML modules written in Perl. The basic interface for YAML consists of two functions: "Dump" and "Load". The real work is done by the modules YAML::Dumper and YAML::Loader.

Different YAML module distributions can be created by subclassing YAML.pm and YAML::Loader and YAML::Dumper. For example, YAML-Simple consists of YAML::Simple YAML::Dumper::Simple and YAML::Loader::Simple.

Why would there be more than one implementation of YAML? Well, despite YAML's offering of being a simple data format, YAML is actually very deep and complex. Implementing the entirety of the YAML specification is a daunting task.

For this reason I am currently working on 3 different YAML implementations.

YAML

The main YAML distribution will keeping evolving to support the entire YAML specification in pure Perl. This may not be the fastest or most stable module though. Currently, YAML.pm has lots of known bugs. It is mostly a great tool for dumping Perl data structures to a readable form.

YAML::Tiny

The point of YAML::Tiny is to strip YAML down to the 90% that people use most and offer that in a small, fast, stable, pure Perl form. YAML::Tiny will simply die when it is asked to do something it can't.

YAML::Syck

"libsyck" is the C based YAML processing library used by the Ruby programming language (and also Python, PHP and Pugs). YAML::Syck is the Perl binding to "libsyck". It should be very fast, but may have problems of its own. It will also require C compilation.

NOTE: Audrey Tang has actually completed this module and it works great and is 10 times faster than YAML.pm.

In the future, there will likely be even more YAML modules. Remember, people other than Ingy are allowed to write YAML modules!

FUNCTIONAL USAGE

YAML is completely OO under the hood. Still it exports a few useful top level functions so that it is dead simple to use. These functions just do the OO stuff for you. If you want direct access to the OO API see the documentation for YAML::Dumper and YAML::Loader.

Exported Functions

The following functions are exported by YAML.pm by default. The reason they are exported is so that YAML works much like Data::Dumper. If you don't want functions to be imported, just use YAML with an empty import list:

```
use YAML ();
```

Dump(list-of-Perl-data-structures)

Turn Perl data into YAML. This function works very much like Data::Dumper::Dumper(). It takes a list of Perl data structures and dumps them into a serialized form. It returns a string containing the YAML stream. The structures can be references or plain scalars.

Load(string-containing-a-YAML-stream)

Turn YAML into Perl data. This is the opposite of Dump. Just like Storable's thaw() function or the eval() function in relation to Data::Dumper. It parses a string containing a valid YAML stream into a list of Perl data structures.

Exportable Functions

These functions are not exported by default but you can request them in an import list like this:

```
use YAML qw'freeze thaw Bless';
```

freeze() and thaw()

Aliases to Dump() and Load() for Storable fans. This will also allow YAML.pm to be plugged directly into modules like POE.pm, that use the freeze/thaw API for internal serialization.

DumpFile(filepath, list)

Writes the YAML stream to a file instead of just returning a string.

LoadFile(filepath)

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

Bless(perl-node, [yaml-node | class-name])

Associate a normal Perl node, with a yaml node. A yaml node is an object tied to the YAML::Node class. The second argument is either a yaml node that you've already created or a class (package) name that supports a "yaml_dump()" function. A "yaml_dump()" function should take a perl node and return a yaml node. If no second argument is provided, Bless will create a yaml node. This node is not returned, but can be retrieved with the Blessed() function.

Here's an example of how to use Bless. Say you have a hash containing three keys, but you only want to dump two of them. Furthermore the keys must be dumped in a certain order. Here's how you do that:

```
use YAML qw(Dump Bless);
$hash = {apple => 'good', banana => 'bad', cauliflower => 'ugly'};
print Dump $hash;
Bless($hash)->keys(['banana', 'apple']);
print Dump $hash;
```

produces:

```
---
apple: good
banana: bad
cauliflower: ugly
---
banana: bad
apple: good
```

Bless returns the tied part of a yaml-node, so that you can call the YAML::Node methods. This is the same thing that YAML::Node::ynode() returns. So another way to do the above example is:

```
use YAML qw(Dump Bless);
use YAML::Node;
$hash = {apple => 'good', banana => 'bad', cauliflower => 'ugly'};
print Dump $hash;
Bless($hash);
$ynode = ynode(Blessed($hash));
$ynode->keys(['banana', 'apple']);
print Dump $hash;
```

Note that Blessing a Perl data structure does not change it anyway. The extra information is stored separately and looked up by the Blessed node's memory address.

Blessed(perl-node)

Returns the yaml node that a particular perl node is associated with (see above). Returns undef if the node is not (YAML) Blessed.

YAML options are set using a group of global variables in the YAML namespace. This is similar to how Data::Dumper works.

For example, to change the indentation width, do something like:

```
local $YAML::Indent = 3;
```

The current options are:

DumperClass

You can override which module/class YAML uses for Dumping data.

LoadBlessed (since 1.25)

Default is undef (false)

The default was changed in version 1.30.

When set to true, YAML nodes with special tags will be automatocally blessed into objects:

```
- !perl/hash:Foo::Bar
  foo: 42
```

When loading untrusted YAML, you should disable this option by setting it to 0. This will also disable setting typeglobs when loading them.

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>

LoaderClass

You can override which module/class YAML uses for Loading data.

Indent

This is the number of space characters to use for each indentation level when doing a Dump(). The default is 2.

By the way, YAML can use any number of characters for indentation at any level. So if you are editing YAML by hand feel free to do it anyway that looks pleasing to you; just be consistent for a given level.

SortKeys

Default is 1. (true)

Tells YAML.pm whether or not to sort hash keys when storing a document.

YAML::Node objects can have their own sort order, which is usually what you want. To override the YAML::Node order and sort the keys anyway, set SortKeys to 2.

Stringify

Default is 0. (false)

Objects with string overloading should honor the overloading and dump the stringification of themselves, rather than the actual object's guts.

Numify

Default is 0. (false)

Values that look like numbers (integers, floats) will be numified when loaded.

UseHeader

Default is 1. (true)

This tells YAML.pm whether to use a separator string for a Dump operation. This only applies to the first document in a stream. Subsequent documents must have a YAML header by definition.

UseVersion

Default is 0. (false)

Tells YAML.pm whether to include the YAML version on the separator/header.

```
--- %YAML:1.0
```

AnchorPrefix

Default is ''.

Anchor names are normally numeric. YAML.pm simply starts with '1' and increases by one for each new anchor. This option allows you to specify a string to be prepended to each anchor number.

UseCode

Setting the UseCode option is a shortcut to set both the DumpCode and LoadCode options at once. Setting UseCode to '1' tells YAML.pm to dump Perl code references as Perl (using B::Deparse) and to load them back into memory using eval(). The reason this has to be an option is that using eval() to parse untrusted code is, well, untrustworthy.

DumpCode

Determines if and how YAML.pm should serialize Perl code references. By default YAML.pm will dump code references as dummy placeholders (much like Data::Dumper). If DumpCode is set to '1' or 'deparse', code references will be dumped as actual Perl code.

LoadCode

LoadCode is the opposite of DumpCode. It tells YAML if and how to deserialize code

references. When set to '1' or 'deparse' it will use "eval()". Since this is
potentially risky, only use this option if you know where your YAML has been.
LoadCode must be enabled also to use the feature of evaluating typeglobs (because with
the typeglob feature you would be able to set the variable $YAML::LoadCode from a YAML
file).

Preserve

When set to true, this option tells the Loader to load hashes into YAML::Node objects.
These are tied hashes. This has the effect of remembering the key order, thus it will
be preserved when the hash is dumped again. See YAML::Node for more information.

UseBlock

YAML.pm uses heuristics to guess which scalar style is best for a given node.
Sometimes you'll want all multiline scalars to use the 'block' style. If so, set this
option to 1.
NOTE: YAML's block style is akin to Perl's here-document.

UseFold (Not supported anymore since v0.60)

If you want to force YAML to use the 'folded' style for all multiline scalars, then
set $UseFold to 1.
NOTE: YAML's folded style is akin to the way HTML folds text, except smarter.

UseAliases

YAML has an alias mechanism such that any given structure in memory gets serialized
once. Any other references to that structure are serialized only as alias markers.
This is how YAML can serialize duplicate and recursive structures.
Sometimes, when you KNOW that your data is nonrecursive in nature, you may want to
serialize such that every node is expressed in full. (ie as a copy of the original).
Setting $YAML::UseAliases to 0 will allow you to do this. This also may result in
faster processing because the lookup overhead is by bypassed.
THIS OPTION CAN BE DANGEROUS. If your data is recursive, this option will cause Dump()
to run in an endless loop, chewing up your computers memory. You have been warned.

CompressSeries

Default is 1.
Compresses the formatting of arrays of hashes:

```
-
  foo: bar
```

```
-
  bar: foo
```

becomes:

```
- foo: bar
- bar: foo
```

Since this output is usually more desirable, this option is turned on by default.

QuoteNumericStrings

Default is 0. (false)

Adds detection mechanisms to encode strings that resemble numbers with mandatory quoting.

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

YAML TERMINOLOGY

YAML is a full featured data serialization language, and thus has its own terminology.

It is important to remember that although YAML is heavily influenced by Perl and Python, it is a language in its own right, not merely just a representation of Perl structures.

YAML has three constructs that are conspicuously similar to Perl's hash, array, and scalar. They are called mapping, sequence, and string respectively. By default, they do what you would expect. But each instance may have an explicit or implicit tag (type) that makes it behave differently. In this manner, YAML can be extended to represent Perl's Glob or Python's tuple, or Ruby's Bigint.

stream

A YAML stream is the full sequence of Unicode characters that a YAML parser would read or a YAML emitter would write. A stream may contain one or more YAML documents separated by YAML headers.

```
---
a: mapping
foo: bar
---
- a
- sequence
```

document

A YAML document is an independent data structure representation within a stream. It is

a top level node. Each document in a YAML stream must begin with a YAML header line.

Actually the header is optional on the first document.

```
---
This: top level mapping
is:
  - a
  - YAML
  - document
```

header

A YAML header is a line that begins a YAML document. It consists of three dashes,

possibly followed by more info. Another purpose of the header line is that it serves

as a place to put top level tag and anchor information.

```
--- !recursive-sequence &001
- * 001
- * 001
```

node

A YAML node is the representation of a particular data structure. Nodes may contain

other nodes. (In Perl terms, nodes are like scalars. Strings, arrayrefs and hashrefs.

But this refers to the serialized format, not the in- memory structure.)

tag This is similar to a type. It indicates how a particular YAML node serialization

should be transferred into or out of memory. For instance a Foo::Bar object would use

the tag 'perl/Foo::Bar':

```
- !perl/Foo::Bar
  foo: 42
  bar: stool
```

collection

A collection is the generic term for a YAML data grouping. YAML has two types of

collections: mappings and sequences. (Similar to hashes and arrays)

mapping

A mapping is a YAML collection defined by unordered key/value pairs with unique keys.

By default YAML mappings are loaded into Perl hashes.

```
a mapping:
  foo: bar
```

two: times two is 4

sequence

A sequence is a YAML collection defined by an ordered list of elements. By default YAML sequences are loaded into Perl arrays.

a sequence:

- one bourbon

- one scotch

- one beer

scalar

A scalar is a YAML node that is a single value. By default YAML scalars are loaded into Perl scalars.

a scalar key: a scalar value

YAML has many styles for representing scalars. This is important because varying data will have varying formatting requirements to retain the optimum human readability.

plain scalar

A plain scalar is unquoted. All plain scalars are automatic candidates for "implicit tagging". This means that their tag may be determined automatically by examination. The typical uses for this are plain alpha strings, integers, real numbers, dates, times and currency.

- a plain string

- -42

- 3.1415

- 12:34

- 123 this is an error

single quoted scalar

This is similar to Perl's use of single quotes. It means no escaping except for single quotes which are escaped by using two adjacent single quotes.

- 'When I say ''\n'' I mean "backslash en"'

double quoted scalar

This is similar to Perl's use of double quotes. Character escaping can be used.

- "This scalar\nhas two lines, and a bell -->\a"

folded scalar

This is a multiline scalar which begins on the next line. It is indicated by a single

right angle bracket. It is unescaped like the single quoted scalar. Line folding is also performed.

```
- >
  This is a multiline scalar which begins on
  the next line. It is indicated by a single
  carat. It is unescaped like the single
  quoted scalar. Line folding is also
  performed.
```

block scalar

This final multiline form is akin to Perl's here-document except that (as in all YAML data) scope is indicated by indentation. Therefore, no ending marker is required. The data is verbatim. No line folding.

```
- |
  QTY  DESC         PRICE  TOTAL
  ---  ----         -----  -----
    1  Foo Fighters  $19.95 $19.95
    2  Bar Belles    $29.95 $59.90
```

parser

A YAML processor has four stages: parse, load, dump, emit.

A parser parses a YAML stream. YAML.pm's Load() function contains a parser.

loader

The other half of the Load() function is a loader. This takes the information from the parser and loads it into a Perl data structure.

dumper

The Dump() function consists of a dumper and an emitter. The dumper walks through each Perl data structure and gives info to the emitter.

emitter

The emitter takes info from the dumper and turns it into a YAML stream.

NOTE: In YAML.pm the parserloader and the dumperemitter code are currently very closely tied together. In the future they may be broken into separate stages.

For more information please refer to the immensely helpful YAML specification available at <http://www.yaml.org/spec/>.

YSH - THE YAML SHELL

libyaml-shell-perl contains a script called 'ysh', the YAML shell. ysh provides a simple, interactive way to play with YAML. If you type in Perl code, it displays the result in YAML. If you type in YAML it turns it into Perl code.

To run ysh, (assuming you installed it along with YAML.pm) simply type:

```
ysh [options]
```

Please read the "ysh" documentation for the full details. There are lots of options.

BUGS & DEFICIENCIES

If you find a bug in YAML, please try to recreate it in the YAML Shell with logging turned on ('ysh -L'). When you have successfully reproduced the bug, please mail the LOG file to the author (ingy@cpan.org).

WARNING: This is still ALPHA code. Well, most of this code has been around for years...

BIGGER WARNING: YAML.pm has been slow in the making, but I am committed to having top notch YAML tools in the Perl world. The YAML team is close to finalizing the YAML 1.1 spec. This version of YAML.pm is based off of a very old pre 1.0 spec. In actuality there isn't a ton of difference, and this YAML.pm is still fairly useful. Things will get much better in the future.

RESOURCES

<http://lists.sourceforge.net/lists/listinfo/yaml-core> is the mailing list. This is where the language is discussed and designed.

<http://www.yaml.org> is the official YAML website.

<http://www.yaml.org/spec/> is the YAML 1.2 specification.

<http://yaml.kwiki.org> is the official YAML wiki.

SEE ALSO

? YAML::XS

AUTHOR

Ingy d?t Net <ingy@cpan.org>

COPYRIGHT AND LICENSE