Yaml Cookbook

at [the YamlForRuby site]

Welcome to the Yaml Cookbook for Ruby. This version of the Yaml Cookbook focuses on the Ruby implementation of Yaml by comparing Yaml documents with their Ruby counterparts.

YAML(tm) is a readable text format for data structures. As you'll see below, YAML can handle many common data types and structures. And what YAML can't handle natively can be supported through flexible type families. For example, YAML for Ruby uses type families to support storage of regular expressions, ranges and object instances.

You can learn more about YAML at YAML.org or the YAML WikiWiki.

Collections

Simple Sequence

Brief

You can specify a list in YAML by placing each member of the list on a new line with an opening dash. These lists are called sequences.

Yaml

```
Simple Sequence   in YAML?
```

```
- apple
- banana
- carrot
```

Ruby

```
Simple Sequence   in Ruby?
```

```
['apple', 'banana', 'carrot']
```

Nested Sequences

Brief

You can include a sequence within another sequence by giving the sequence an empty dash, followed by an indented list.

Yaml

```
Nested Sequences   in YAML?
```

```
-
 - foo
 - bar
 - baz
```

Ruby

```
Nested Sequences   in Ruby?
```

```
[['foo', 'bar', 'baz']]
```

Mixed Sequences

Brief

Sequences can contain any YAML data, including strings and other sequences.

Yaml

```
Mixed Sequences   in YAML?
```

```
- apple
-
 - foo
 - bar
 - x123
- banana
- carrot
```

Ruby

```
Mixed Sequences   in Ruby?
```

```
['apple', ['foo', 'bar', 'x123'], 'banana', 'carrot']
```

Deeply Nested Sequences

Brief

Sequences can be nested even deeper, with each level of indentation representing a level of depth.

Yaml

```
Deeply Nested Sequences   in YAML?
```

```
-
 -
  - uno
  - dos
```

Ruby

Deeply Nested Sequences in Ruby?

```
[[['uno', 'dos']]]
```

Simple Mapping

Brief

You can add a keyed list (also known as a dictionary or hash) to your document by placing each member of the list on a new line, with a colon seperating the key from its value. In YAML, this type of list is called a mapping.

Yaml

Simple Mapping in YAML?

```
foo: whatever
bar: stuff
```

Ruby

Simple Mapping in Ruby?

```
{ 'foo' => 'whatever', 'bar' => 'stuff' }
```

Sequence in a Mapping

Brief

A value in a mapping can be a sequence.

Yaml

Sequence in a Mapping in YAML?

```
foo: whatever
bar:
 - uno
 - dos
```

Ruby

Sequence in a Mapping in Ruby?

```
{ 'foo' => 'whatever', 'bar' => [ 'uno', 'dos' ] }
```

Nested Mappings

Brief

A value in a mapping can be another mapping.

Yaml

Nested Mappings in YAML?

```
foo: whatever
bar:
 fruit: apple
 name: steve
 sport: baseball
```

Ruby

Nested Mappings in Ruby?

```
{ 'foo' => 'whatever',
  'bar' => {
     'fruit' => 'apple',
     'name' => 'steve',
     'sport' => 'baseball'
```

```
    }
}
```

Mixed Mapping

Brief

A mapping can contain any assortment of mappings and sequences as values.

Yaml

```
Mixed Mapping   in YAML?
```

```
foo: whatever
bar:
 -
   fruit: apple
   name: steve
   sport: baseball
 - more
 -
   python: rocks
   perl: papers
   ruby: scissorses
```

Ruby

```
Mixed Mapping   in Ruby?
```

```
{ 'foo' => 'whatever',
  'bar' => [
    {
        'fruit' => 'apple',
        'name' => 'steve',
        'sport' => 'baseball'
    },
    'more',
    {
        'python' => 'rocks',
        'perl' => 'papers',
        'ruby' => 'scissorses'
    }
  ]
}
```

Mapping-in-Sequence Shortcut

Brief

If you are adding a mapping to a sequence, you can place the mapping on the same line as the dash as a shortcut.

Yaml

```
Mapping-in-Sequence Shortcut   in YAML?
```

```
- work on YAML.py:
   - work on Store
```

Ruby

```
Mapping-in-Sequence Shortcut   in Ruby?
```

```
[ { 'work on YAML.py' => ['work on Store'] } ]
```

Sequence-in-Mapping Shortcut

Brief

The dash in a sequence counts as indentation, so you can add a sequence inside of a mapping without needing spaces as indentation.

Yaml

```
Sequence-in-Mapping Shortcut   in YAML?
```

```
allow:
- 'localhost'
- '%.sourceforge.net'
- '%.freepan.org'
```

Ruby

```
Sequence-in-Mapping Shortcut   in Ruby?
{ 'allow' => [ 'localhost', '%.sourceforge.net', '%.freepan.org' ] }
```

Merge key

Brief

A merge key ('<<') can be used in a mapping to insert other mappings. If the value associated with the merge key is a mapping, each of its key/value pairs is inserted into the current mapping.

Yaml

```
Merge key   in YAML?
mapping:
  name: Joe
  job: Accountant
  <<:
    age: 38
```

Ruby

```
Merge key   in Ruby?
{ 'mapping' =>
  { 'name' => 'Joe',
    'job' => 'Accountant',
    'age' => 38
  }
}
```

Inline Collections

Simple Inline Array

Brief

Sequences can be contained on a single line, using the inline syntax. Separate each entry with commas and enclose in square brackets.

Yaml

```
Simple Inline Array   in YAML?
---
seq: [ a, b, c ]
```

Ruby

```
Simple Inline Array   in Ruby?
{ 'seq' => [ 'a', 'b', 'c' ] }
```

Simple Inline Hash

Brief

Mapping can also be contained on a single line, using the inline syntax. Each key-value pair is separated by a colon, with a comma between each entry in the mapping. Enclose with curly braces.

Yaml

```
Simple Inline Hash   in YAML?
```

```
---
hash: { name: Steve, foo: bar }
```

Ruby

```
Simple Inline Hash   in Ruby?
{ 'hash' => { 'name' => 'Steve', 'foo' => 'bar' } }
```

Multi-line Inline Collections

Brief

Both inline sequences and inline mappings can span multiple lines, provided that you indent the additional lines.

Yaml

```
 Multi-
line Inline Collections          in YAML?
 languages: [ Ruby,
              Perl,
              Python ]
 websites: { YAML: yaml.org,
             Ruby: ruby-lang.org,
             Python: python.org,
             Perl: use.perl.org }
```

Ruby

```
 Multi-
line Inline Collections          in Ruby?
 { 'languages' => [ 'Ruby', 'Perl', 'Python' ],
   'websites' => {
     'YAML' => 'yaml.org',
     'Ruby' => 'ruby-lang.org',
     'Python' => 'python.org',
     'Perl' => 'use.perl.org'
   }
 }
```

Commas in Values

Brief

List items in collections are delimited by commas, but there must be a space after each comma. This allows you to add numbers without quoting.

Yaml

```
Commas in Values   in YAML?
attendances: [ 45,123, 70,000, 17,222 ]
```

Ruby

```
Commas in Values   in Ruby?
{ 'attendances' => [ 45123, 70000, 17222 ] }
```

Basic Types

Strings

Brief

Any group of characters beginning with an alphabetic or numeric character is a string, unless it belongs to one of the groups below (such as an Integer or Time).

Yaml

```
Strings   in YAML?
```

```
--- String
```

Ruby

```
Strings   in Ruby?
```

```
'String'
```

String characters

Brief

A string can contain any alphabetic or numeric character, along with many punctuation characters, including the period, dash, space, quotes, exclamation, and question mark.

Yaml

```
String characters   in YAML?
```

```
- What's Yaml?
- It's for writing data structures in plain text.
- And?
- And what? That's not good enough for you?
- No, I mean, "And what about Yaml?"
- Oh, oh yeah. Uh.. Yaml for Ruby.
```

Ruby

```
String characters   in Ruby?
```

```
[
  "What's Yaml?",
  "It's for writing data structures in plain text.",
  "And?",
  "And what? That's not good enough for you?",
  "No, I mean, \"And what about Yaml?\"",
  "Oh, oh yeah. Uh.. Yaml for Ruby."
]
```

Indicators in Strings

Brief

Be careful using indicators in strings. In particular, the comma, colon, and pound sign must be used carefully.

Yaml

```
Indicators in Strings   in YAML?
```

```
the colon followed by space is an indicator: but is a string:right here
same for the pound sign: here we have it#in a string
the comma can, honestly, be used in most cases: [ but not in, inline collections ]
```

Ruby

```
Indicators in Strings   in Ruby?
```

```
{
  'the colon followed by space is an indicator' => 'but is a string:right here',
  'same for the pound sign' => 'here we have it#in a string',
  'the comma can, honestly, be used in most cases' => [ 'but not in', 'inline collections' ]
}
```

Forcing Strings

Brief

Any YAML type can be forced into a string using the explicit !str method.

Yaml

Forcing Strings in YAML?

```
date string: !str 2001-08-01
number string: !str 192
```

Ruby

Forcing Strings in Ruby?

```
{
  'date string' => '2001-08-01',
  'number string' => '192'
}
```

Single-quoted Strings

Brief

You can also enclose your strings within single quotes, which allows use of slashes, colons, and other indicators freely. Inside single quotes, you can represent a single quote in your string by using two single quotes next to each other.

Yaml

Single-quoted Strings in YAML?

```
all my favorite symbols: '#:!/%.)'
a few i hate: '&(*'
why do i hate them?: 'it''s very hard to explain'
```

Ruby

Single-quoted Strings in Ruby?

```
{
  'all my favorite symbols' => '#:!/%.)',
  'a few i hate' => '&(*',
  'why do i hate them?' => 'it\'s very hard to explain'
}
```

Double-quoted Strings

Brief

Enclosing strings in double quotes allows you to use escapings to represent ASCII and Unicode characters.

Yaml

Double-quoted Strings in YAML?

```
i know where i want my line breaks: "one here\nand another here\n"
```

Ruby

Double-quoted Strings in Ruby?

```
{
  'i know where i want my line breaks' => "one here\nand another here\n"
}
```

Multi-line Quoted Strings

Brief

Both single- and double-quoted strings may be carried on to new lines in your YAML document. They must be indented a step and indentation is interpreted as a single space.

Yaml

Multi-line Quoted Strings in YAML?

```
i want a long string: "so i'm going to
  let it go on and on to other lines
  until i end it with a quote."
```

Ruby

Multi-line Quoted Strings in Ruby?

```
{ 'i want a long string' => "so i'm going to " +
     "let it go on and on to other lines " +
     "until i end it with a quote."
}
```

Plain scalars

Brief

Unquoted strings may also span multiple lines, if they are free of YAML space indicators and indented.

Yaml

Plain scalars in YAML?

```
- My little toe is broken in two places;
- I'm crazy to have skied this way;
- I'm not the craziest he's seen, since there was always the German guy
  who skied for 3 hours on a broken shin bone (just below the kneecap);
- Nevertheless, second place is respectable, and he doesn't
  recommend going for the record;
- He's going to put my foot in plaster for a month;
- This would impair my skiing ability somewhat for the
  duration, as can be imagined.
```

Ruby

Plain scalars in Ruby?

```
[
  "My little toe is broken in two places;",
  "I'm crazy to have skied this way;",
  "I'm not the craziest he's seen, since there was always " +
     "the German guy who skied for 3 hours on a broken shin " +
     "bone (just below the kneecap);",
  "Nevertheless, second place is respectable, and he doesn't " +
     "recommend going for the record;",
  "He's going to put my foot in plaster for a month;",
  "This would impair my skiing ability somewhat for the duration, " +
     "as can be imagined."
]
```

Null

Brief

You can use the tilde '~' character for a null value.

Yaml

Null in YAML?

```
name: Mr. Show
hosted by: Bob and David
date of next season: ~
```

Ruby

Null in Ruby?

```
{
  'name' => 'Mr. Show',
  'hosted by' => 'Bob and David',
  'date of next season' => nil
}
```

Boolean

Brief

You can use 'true' and 'false' for boolean values.

Yaml

```
Boolean in YAML?

Is Gus a Liar?: true
Do I rely on Gus for Sustenance?: false
```

Ruby

```
Boolean in Ruby?

{
  'Is Gus a Liar?' => true,
  'Do I rely on Gus for Sustenance?' => false
}
```

Integers

Brief

An integer is a series of numbers, optionally starting with a positive or negative sign. Integers may also contain commas for readability.

Yaml

```
Integers in YAML?

zero: 0
simple: 12
one-thousand: 1,000
negative one-thousand: -1,000
```

Ruby

```
Integers in Ruby?

{
  'zero' => 0,
  'simple' => 12,
  'one-thousand' => 1000,
  'negative one-thousand' => -1000
}
```

Integers as Map Keys

Brief

An integer can be used a dictionary key.

Yaml

```
Integers as Map Keys in YAML?

1: one
2: two
3: three
```

Ruby

```
Integers as Map Keys in Ruby?

{
    1 => 'one',
    2 => 'two',
    3 => 'three'
}
```

Floats

Brief

Floats are represented by numbers with decimals, allowing for scientific notation, as well as positive and negative infinity and "not a number."

Yaml

Floats in YAML?

```
a simple float: 2.00
larger float: 1,000.09
scientific notation: 1.00009e+3
```

Ruby

Floats in Ruby?

```
{
  'a simple float' => 2.0,
  'larger float' => 1000.09,
  'scientific notation' => 1000.09
}
```

Time

Brief

You can represent timestamps by using ISO8601 format, or a variation which allows spaces between the date, time and time zone.

Yaml

Time in YAML?

```
iso8601: 2001-12-14t21:59:43.10-05:00
space seperated: 2001-12-14 21:59:43.10 -05:00
```

Ruby

Time in Ruby?

```
{
  'iso8601' => YAML::mktime( 2001, 12, 14, 21, 59, 43, 0.10, "-05:00" ),
  'space seperated' => YAML::mktime( 2001, 12, 14, 21, 59, 43, 0.10, "-05:00" )
}
```

Date

Brief

A date can be represented by its year, month and day in ISO8601 order.

Yaml

Date in YAML?

```
--- 1976-07-31
```

Ruby

Date in Ruby?

```
Date.new( 1976, 7, 31 )
```

Blocks

Single ending newline

Brief

A pipe character, followed by an indented block of text is treated as a literal block, in which newlines are preserved throughout the block, including the final newline.

Yaml

Single ending newline in YAML?

```
---
this: |
    Foo
    Bar
```

Ruby

```
Single ending newline   in Ruby?
{ 'this' => "Foo\nBar\n" }
```

The '+' indicator

Brief

The '+' indicator says to keep newlines at the end of text blocks.

Yaml

```
The '+' indicator   in YAML?
normal: |
  extra new lines not kept

preserving: |+
  extra new lines are kept


dummy: value
```

Ruby

```
The '+' indicator   in Ruby?
{
    'normal' => "extra new lines not kept\n",
    'preserving' => "extra new lines are kept\n\n\n",
    'dummy' => 'value'
}
```

Three trailing newlines in literals

Brief

To give you more control over how space is preserved in text blocks, YAML has the keep '+' and chomp '-' indicators. The keep indicator will preserve all ending newlines, while the chomp indicator will strip all ending newlines.

Yaml

```
Three trailing newlines in literals   in YAML?
clipped: |
    This has one newline.



same as "clipped" above: "This has one newline.\n"

stripped: |-
    This has no newline.



same as "stripped" above: "This has no newline."

kept: |+
    This has four newlines.



same as "kept" above: "This has four newlines.\n\n\n\n"
```

Ruby

```
Three trailing newlines in literals   in Ruby?
```

```
{
  'clipped' => "This has one newline.\n",
  'same as "clipped" above' => "This has one newline.\n",
  'stripped' => 'This has no newline.',
  'same as "stripped" above' => 'This has no newline.',
  'kept' => "This has four newlines.\n\n\n\n",
  'same as "kept" above' => "This has four newlines.\n\n\n\n"
}
```

Extra trailing newlines with spaces

Brief

Normally, only a single newline is kept from the end of a literal block, unless the keep '+' character is used in combination with the pipe. The following example will preserve all ending whitespace since the last line of both literal blocks contains spaces which extend past the indentation level.

Yaml

```
Extra trailing newlines with spaces   in YAML?
```

```
---
this: |
    Foo


kept: |+
    Foo
```

Ruby

```
Extra trailing newlines with spaces   in Ruby?
```

```
{ 'this' => "Foo\n\n  \n",
  'kept' => "Foo\n\n  \n" }
```

Folded Block in a Sequence

Brief

A greater-then character, followed by an indented block of text is treated as a folded block, in which lines of text separated by a single newline are concatenated as a single line.

Yaml

```
Folded Block in a Sequence   in YAML?
```

```
---
- apple
- banana
- >
    can't you see
    the beauty of yaml?
    hmm
- dog
```

Ruby

```
Folded Block in a Sequence   in Ruby?
```

```
[
    'apple',
    'banana',
    "can't you see the beauty of yaml? hmm\n",
    'dog'
]
```

Folded Block as a Mapping Value

Brief

Both literal and folded blocks can be used in collections, as values in a sequence or a mapping.

Yaml

Folded Block as a Mapping Value in YAML?

```
---
quote: >
    Mark McGwire's
    year was crippled
    by a knee injury.
source: espn
```

Ruby

Folded Block as a Mapping Value in Ruby?

```
{
    'quote' => "Mark McGwire's year was crippled by a knee injury.\n",
    'source' => 'espn'
}
```

Three trailing newlines in folded blocks

Brief

The keep and chomp indicators can also be applied to folded blocks.

Yaml

Three trailing newlines in folded blocks in YAML?

```
clipped: >
    This has one newline.



same as "clipped" above: "This has one newline.\n"

stripped: >-
    This has no newline.



same as "stripped" above: "This has no newline."

kept: >+
    This has four newlines.



same as "kept" above: "This has four newlines.\n\n\n\n"
```

Ruby

Three trailing newlines in folded blocks in Ruby?

```
{
  'clipped' => "This has one newline.\n",
  'same as "clipped" above' => "This has one newline.\n",
  'stripped' => 'This has no newline.',
  'same as "stripped" above' => 'This has no newline.',
  'kept' => "This has four newlines.\n\n\n\n",
  'same as "kept" above' => "This has four newlines.\n\n\n\n"
}
```

Aliases and Anchors

Simple Alias Example

Brief

If you need to refer to the same item of data twice, you can give that item an alias. The alias is a plain string, starting with an ampersand. The item may then be referred to by the alias throughout your document by using an asterisk before the name of the alias. This is called an anchor.

Yaml

Simple Alias Example in YAML?

```
- &showell Steve
- Clark
- Brian
- Oren
- *showell
```

Ruby

Simple Alias Example in Ruby?

```
showell = 'Steve'

[ showell, 'Clark', 'Brian', 'Oren', showell ]
```

Alias of a Mapping

Brief

An alias can be used on any item of data, including sequences, mappings, and other complex data types.

Yaml

Alias of a Mapping in YAML?

```
- &hello
    Meat: pork
    Starch: potato
- banana
- *hello
```

Ruby

Alias of a Mapping in Ruby?

```
hello = { 'Meat' => 'pork', 'Starch' => 'potato' }

[
  hello,
  'banana',
  hello
]
```

Documents

Trailing Document Separator

Brief

You can separate YAML documents with a string of three dashes.

Yaml

Trailing Document Separator in YAML?

```
- foo: 1
  bar: 2
---
more: stuff
```

Ruby

Trailing Document Separator in Ruby?

```
[ { 'foo' => 1, 'bar' => 2 } ]
```

Leading Document Separator

Brief

You can explicity give an opening document separator to your YAML stream.

Yaml

```
Leading Document Separator   in YAML?
```

```
---
- foo: 1
  bar: 2
---
more: stuff
```

Ruby

```
Leading Document Separator   in Ruby?
```

```
[ { 'foo' => 1, 'bar' => 2 } ]
```

YAML Header

Brief

The opening separator can contain directives to the YAML parser, such as the version number.

Yaml

```
YAML Header   in YAML?
```

```
--- %YAML:1.0
foo: 1
bar: 2
```

Ruby

```
YAML Header   in Ruby?
```

```
y = Stream.new
y.add( { 'foo' => 1, 'bar' => 2 } )
```

Red Herring Document Separator

Brief

Separators included in blocks or strings are treated as blocks or strings, as the document separator should have no indentation preceding it.

Yaml

```
Red Herring Document Separator   in YAML?
```

```
foo: |
    ---
```

Ruby

```
Red Herring Document Separator   in Ruby?
```

```
{ 'foo' => "---\n" }
```

Multiple Document Separators in Block

Brief

This technique allows you to embed other YAML documents within literal blocks.

Yaml

```
Multiple Document Separators in Block   in YAML?
```

```
foo: |
    ---
    foo: bar
    ---
    yo: baz
```

```
bar: |
    fooness
```

Ruby

Multiple Document Separators in Block in Ruby?

```
{
   'foo' => "---\nfoo: bar\n---\nyo: baz\n",
   'bar' => "fooness\n"
}
```

YAML For Ruby

Symbols

Brief

Ruby Symbols can be simply serialized using the !ruby/symbol transfer method, or the abbreviated !ruby/sym.

Yaml

Symbols in YAML?

```
simple symbol: !ruby/symbol Simple
shortcut syntax: !ruby/sym Simple
symbols in seqs:
  - !ruby/symbol ValOne
  - !ruby/symbol ValTwo
  - !ruby/symbol ValThree
symbols in maps:
  - !ruby/symbol MapKey: !ruby/symbol MapValue
```

Ruby

Symbols in Ruby?

```
{ 'simple symbol' => :Simple,
  'shortcut syntax' => :Simple,
  'symbols in seqs' => [ :ValOne, :ValTwo, :ValThree ],
  'symbols in maps' => [ { :MapKey => :MapValue } ]
}
```

Ranges

Brief

Ranges are serialized with the !ruby/range type family.

Yaml

Ranges in YAML?

```
normal range: !ruby/range 10..20
exclusive range: !ruby/range 11...20
negative range: !ruby/range -1..-5
? !ruby/range 0..40
: range as a map key
```

Ruby

Ranges in Ruby?

```
{ 'normal range' => (10..20),
  'exclusive range' => (11...20),
  'negative range' => (-1..-5),
  (0..40) => 'range as a map key'
}
```

Regexps

Brief

Regexps may be serialized to YAML, both its syntax and any modifiers.

Yaml

Regexps in YAML?

```
case-insensitive: !ruby/regexp "/George McFly/i"
complex: !ruby/regexp "/\\A\"((?:[^\"]|\\\")+)\"/"
simple: !ruby/regexp '/a.b/'
```

Ruby

Regexps in Ruby?

```
{ 'simple' => /a.b/, 'complex' => /\A"((?:[^"]|\")+)"/,
  'case-insensitive' => /George McFly/i }
```

Perl Regexps

Brief

Regexps may also be imported from serialized Perl.

Yaml

Perl Regexps in YAML?

```
--- !perl/regexp:
  REGEXP: "R[Uu][Bb][Yy]$"
  MODIFIERS: i
```

Ruby

Perl Regexps in Ruby?

```
/R[Uu][Bb][Yy]$/i
```

Struct class

Brief

The Ruby Struct class is registered as a YAML builtin type through Ruby, so it can safely be serialized. To use it, first make sure you define your Struct with Struct::new. Then, you are able to serialize with Struct#to_yaml and unserialize from a YAML stream.

Yaml

Struct class in YAML?

```
--- !ruby/struct:BookStruct
  author: Yukihiro Matsumoto
  title: Ruby in a Nutshell
  year: 2002
  isbn: 0-596-00214-9
```

Ruby

Struct class in Ruby?

```
book_struct = Struct::new( "BookStruct", :author, :title, :year, :isbn )

book_struct.new( "Yukihiro Matsumoto", "Ruby in a Nutshell", 2002, "0-596-00214-9" )
```

Nested Structs

Brief

As with other YAML builtins, you may nest the Struct inside of other Structs or other data types.

Yaml

Nested Structs in YAML?

```
- !ruby/struct:FoodStruct
  name: Nachos
  ingredients:
    - Mission Chips
    - !ruby/struct:FoodStruct
      name: Tostitos Nacho Cheese
      ingredients:
        - Milk and Enzymes
        - Jack Cheese
        - Some Volatile Chemicals
      taste: Angelic
    - Sour Cream
  taste: Zesty
- !ruby/struct:FoodStruct
  name: Banana Cream Pie
  ingredients:
    - Bananas
    - Creamy Stuff
    - And Such
  taste: Puffy
```

Ruby

Nested Structs in Ruby?

```
food_struct = Struct::new( "FoodStruct", :name, :ingredients, :taste )

[
  food_struct.new( 'Nachos', [ 'Mission Chips',
    food_struct.new( 'Tostitos Nacho Cheese', [ 'Milk and Enzymes', 'Jack Cheese', 'Some Volatile Chemicals' ], 'An
    'Sour Cream' ], 'Zesty' ),
  food_struct.new( 'Banana Cream Pie', [ 'Bananas', 'Creamy Stuff', 'And Such' ], 'Puffy' )
]
```

Objects

Brief

YAML has generic support for serializing objects from any class available in Ruby. If using the generic object serialization, no extra code is needed.

Yaml

Objects in YAML?

```
--- !ruby/object:YAML::Zoolander
  name: Derek
  look: Blue Steel
```

Ruby

Objects in Ruby?

```
class Zoolander
  attr_accessor :name, :look
  def initialize( look )
    @name = "Derek"
    @look = look
  end
  def ==( z )
    self.name == z.name and self.look == z.look
  end
end

Zoolander.new( "Blue Steel" )
```

Extending Kernel::Array

Brief

When extending the Array class, your instances of such a class will dump as YAML sequences, tagged with a class name.

Yaml

Extending Kernel::Array in YAML?

```
--- !ruby/array:YAML::MyArray
- jacket
- sweater
- windbreaker
```

Ruby

Extending Kernel::Array in Ruby?

```
class MyArray < Kernel::Array; end

outerwear = MyArray.new
outerwear << 'jacket'
outerwear << 'sweater'
outerwear << 'windbreaker'
outerwear
```

Extending Kernel::Hash

Brief

When extending the Hash class, your instances of such a class will dump as YAML maps, tagged with a class name.

Yaml

Extending Kernel::Hash in YAML?

```
--- !ruby/hash:YAML::MyHash
Black Francis: Frank Black
Kim Deal: Breeders
Joey Santiago: Martinis
```

Ruby

Extending Kernel::Hash in Ruby?

```
# Note that the @me attribute isn't dumped
# because the default to_yaml is trained
# to dump as a regular Hash.
class MyHash < Kernel::Hash
  attr_accessor :me
  def initialize
    @me = "Why"
  end
end

pixies = MyHash.new
pixies['Black Francis'] = 'Frank Black'
pixies['Kim Deal'] = 'Breeders'
pixies['Joey Santiago'] = 'Martinis'
pixies
```