Home

Pages Classes Methods

Search

Table of Contents

Show/hide navigation

Parent

Object

Methods

::new
::version
#def_class
#def_method
#def_module
#location=
#make_compiler
#new_toplevel
#result
#result_with_hash
#run
#set_eoutvar

class ERB

ERB – Ruby Templating

Introduction

ERB provides an easy to use but powerful templating system for Ruby. Using ERB, actual Ruby code can be added to any plain text document for the purposes of generating document information details and/or flow control.

A very simple example is this:

```
require 'erb'

x = 42
template = ERB.new <<-EOF
  The value of x is: <%= x %>
EOF
puts template.result(binding)
```

Prints: The value of x is: 42

More complex examples are given below.

Recognized Tags

ERB recognizes certain tags in the provided template and converts them based on the rules below:

```
<% Ruby code -- inline with output %>
<%= Ruby expression -- replace with result %>
<%# comment -- ignored -- useful in testing %> (`<% #` doesn't work. Don't us
% a line of Ruby code -- treated as <% line %> (optional -- see ERB.new)
%% replaced with % if first thing on a line and % processing is used
<%% or %%> -- replace with <% or %> respectively
```

All other text is passed through ERB filtering unchanged.

Options

There are several settings you can change when you use ERB:

- the nature of the tags that are recognized;

- the binding used to resolve local variables in the template.

See the `ERB.new` and `ERB#result` methods for more detail.

Character encodings

ERB (or Ruby code generated by ERB) returns a string in the same character encoding as the input string. When the input string has a magic comment, however, it returns a string in the encoding specified by the magic comment.

```
# -*- coding: utf-8 -*-
require 'erb'

template = ERB.new <<EOF
<%#-*- coding: Big5 -*-%>
  \_\_ENCODING\_\_ is <%= \_\_ENCODING\_\_ %>.
EOF
puts template.result
```

*Prints: _ENCODING_* is Big5.

Examples

Plain Text

ERB is useful for any generic templating situation. Note that in this example, we use the convenient "% at start of line" tag, and we quote the template literally with `%q{...}` to avoid trouble with the backslash.

```
require "erb"

# Create template.
template = %q{
  From:  James Edward Gray II <james@grayproductions.net>
  To:  <%= to %>
  Subject:  Addressing Needs

  <%= to[/\w+/] %>:

  Just wanted to send a quick note assuring that your needs are being
  addressed.

  I want you to know that my team will keep working on the issues,
  especially:

  <%# ignore numerous minor requests -- focus on priorities %>
  % priorities.each do |priority|
    * <%= priority %>
  % end

  Thanks for your patience.
  James Edward Gray II
}.gsub(/^  /, '')

message = ERB.new(template, trim_mode: "%<>")

# Set up template data.
to = "Community Spokesman <spokesman@ruby_community.org>"
priorities = [ "Run Ruby Quiz",
               "Document Modules",
               "Answer Questions on Ruby Talk" ]

# Produce result.
email = message.result
puts email
```

Generates:

```
From:  James Edward Gray II <james@grayproductions.net>
To:  Community Spokesman <spokesman@ruby_community.org>
Subject:  Addressing Needs

Community:

Just wanted to send a quick note assuring that your needs are being addressed

I want you to know that my team will keep working on the issues, especially:

    * Run Ruby Quiz
    * Document Modules
    * Answer Questions on Ruby Talk

Thanks for your patience.

James Edward Gray II
```

Ruby in HTML

ERB is often used in `.rhtml` files (HTML with embedded Ruby). Notice the need in this example to provide a special binding when the template is run, so that the instance variables in the Product object can be resolved.

```
require "erb"

# Build template data class.
class Product
  def initialize( code, name, desc, cost )
    @code = code
    @name = name
    @desc = desc
    @cost = cost

    @features = [ ]
  end

  def add_feature( feature )
```

```
    @features << feature
  end

  # Support templating of member data.
  def get_binding
    binding
  end

  # ...
end

# Create template.
template = %{
  <html>
    <head><title>Ruby Toys -- <%= @name %></title></head>
    <body>

      <h1><%= @name %> (<%= @code %>)</h1>
      <p><%= @desc %></p>

      <ul>
        <% @features.each do |f| %>
          <li><b><%= f %></b></li>
        <% end %>
      </ul>

      <p>
        <% if @cost < 10 %>
          <b>Only <%= @cost %>!!!</b>
        <% else %>
           Call for a price, today!
        <% end %>
      </p>

    </body>
  </html>
}.gsub(/^  /, '')

rhtml = ERB.new(template)

# Set up template data.
toy = Product.new( "TZ-1002",
                   "Rubysapien",
                   "Geek's Best Friend!  Responds to Ruby commands...",
                   999.95 )
toy.add_feature("Listens for verbal commands in the Ruby language!")
toy.add_feature("Ignores Perl, Java, and all C variants.")
toy.add_feature("Karate-Chop Action!!!")
toy.add_feature("Matz signature on left leg.")
toy.add_feature("Gem studded eyes... Rubies, of course!")

# Produce result.
rhtml.run(toy.get_binding)
```

Generates (some blank lines removed):

```
<html>
  <head><title>Ruby Toys -- Rubysapien</title></head>
  <body>

    <h1>Rubysapien (TZ-1002)</h1>
    <p>Geek's Best Friend!  Responds to Ruby commands...</p>
    <ul>
        <li><b>Listens for verbal commands in the Ruby language!</b></li>
        <li><b>Ignores Perl, Java, and all C variants.</b></li>
        <li><b>Karate-Chop Action!!!</b></li>
        <li><b>Matz signature on left leg.</b></li>
        <li><b>Gem studded eyes... Rubies, of course!</b></li>
    </ul>

    <p>
      Call for a price, today!
    </p>

  </body>
</html>
```

Notes

There are a variety of templating solutions available in various Ruby projects. For example, RDoc, distributed with Ruby, uses its own template engine, which can be reused elsewhere.

Other popular engines could be found in the corresponding Category of The Ruby Toolbox.

Constants

NOT_GIVEN

VERSION

Attributes

encoding [R]

The encoding to eval

filename [RW]

The optional *filename* argument passed to Kernel#eval when the ERB code is run

lineno [RW]

The optional *lineno* argument passed to Kernel#eval when the ERB code is run

src [R]

The Ruby code generated by ERB

Public Class Methods

new(str, safe_level=NOT_GIVEN, legacy_trim_mode=NOT_GIVEN, legacy_eoutvar=NOT_GIVEN, trim_mode: nil, eoutvar: '_erbout')

Constructs a new ERB object with the template specified in *str*.

An ERB object works by building a chunk of Ruby code that will output the completed template when run.

If *trim_mode* is passed a String containing one or more of the following modifiers, ERB will adjust its code generation as listed:

```
%  enables Ruby code processing for lines beginning with %
<> omit newline for lines starting with <% and ending in %>
>  omit newline for lines ending in %>
-  omit blank lines ending in -%>
```

eoutvar can be used to set the name of the variable ERB will build up its output in. This is useful when you need to run multiple ERB templates through the same binding and/or when you want to control where output ends up. Pass the name of the variable to be used inside a String.

Example

```
require "erb"

# build data class
class Listings
  PRODUCT = { :name => "Chicken Fried Steak",
              :desc => "A well messages pattie, breaded and fried.",
              :cost => 9.95 }

  attr_reader :product, :price

  def initialize( product = "", price = "" )
    @product = product
    @price = price
  end

  def build
    b = binding
    # create and run templates, filling member data variables
    ERB.new(<<~'END_PRODUCT', trim_mode: "", eoutvar: "@product").result b
      <%= PRODUCT[:name] %>
      <%= PRODUCT[:desc] %>
    END_PRODUCT
    ERB.new(<<~'END_PRICE', trim_mode: "", eoutvar: "@price").result b
```

```
        <%= PRODUCT[:name] %> -- <%= PRODUCT[:cost] %>
        <%= PRODUCT[:desc] %>
      END_PRICE
    end
  end

  # setup template data
  listings = Listings.new
  listings.build

  puts listings.product + "\n" + listings.price
```

Generates

```
Chicken Fried Steak
A well messages pattie, breaded and fried.

Chicken Fried Steak -- 9.95
A well messages pattie, breaded and fried.
```

version()

Returns revision information for the erb.rb module.

Public Instance Methods

def_class(superklass=Object, methodname='result')

Define unnamed class which has *methodname* as instance method, and return it.

example:

```
class MyClass_
  def initialize(arg1, arg2)
    @arg1 = arg1;  @arg2 = arg2
  end
end
filename = 'example.rhtml'  # @arg1 and @arg2 are used in example.rhtml
erb = ERB.new(File.read(filename))
erb.filename = filename
MyClass = erb.def_class(MyClass_, 'render()')
print MyClass.new('foo', 123).render()
```

def_method(mod, methodname, fname='(ERB)')

Define *methodname* as instance method of *mod* from compiled Ruby source.

example:

```
filename = 'example.rhtml'   # 'arg1' and 'arg2' are used in example.rhtml
erb = ERB.new(File.read(filename))
erb.def_method(MyClass, 'render(arg1, arg2)', filename)
print MyClass.new.render('foo', 123)
```

`def_module(methodname='erb')`

Create unnamed module, define *methodname* as instance method of it, and return it.

example:

```
filename = 'example.rhtml'   # 'arg1' and 'arg2' are used in example.rhtml
erb = ERB.new(File.read(filename))
erb.filename = filename
MyModule = erb.def_module('render(arg1, arg2)')
class MyClass
  include MyModule
end
```

`location=((filename, lineno))`

Sets optional filename and line number that will be used in `ERB` code evaluation and error reporting. See also `filename=` and `lineno=`

```
erb = ERB.new('<%= some_x %>')
erb.render
# undefined local variable or method `some_x'
#   from (erb):1

erb.location = ['file.erb', 3]
# All subsequent error reporting would use new location
erb.render
# undefined local variable or method `some_x'
#   from file.erb:4
```

`make_compiler(trim_mode)`

Creates a new compiler for `ERB`. See ERB::Compiler.new for details

`result(b=new_toplevel)`

Executes the generated `ERB` code to produce a completed template, returning the results of that code. (See `ERB::new` for details on how this process can be affected by *safe_level*.)

b accepts a Binding object which is used to set the context of code evaluation.

`result_with_hash(hash)`

Render a template on a new toplevel binding with local variables specified by a Hash object.

`run(b=new_toplevel)`

Generate results and print them. (see `ERB#result`)

`set_eoutvar(compiler, eoutvar = '_erbout')`

Can be used to set *eoutvar* as described in `ERB::new`. It's probably easier to just use the constructor though, since calling this method requires the setup of an `ERB` *compiler* object.

Private Instance Methods

`new_toplevel(vars = nil)`

Returns a new binding each time **near** TOPLEVEL_BINDING for runs that do not specify a binding.