

YARD

Yay! A Ruby Documentation Tool

Yarr, a Ruby Documentation Tool (on Pirate Day)

You Are Riting Docs

<http://yard.soen.ca>

What's a YARD?

“RDoc is Fine.”

YARD is better than fine.

More than HTML

but what do you mean?

WARNING: RDoc Comparison

- RDoc parses code, generates HTML and quits.
- If you need to get at specific data... you can't.
- YARD stores all the data it needs to generate HTML (XML, etc.) in a marshal file, for whenever you need it, however you need it.

Simple Problem:

“I need to export my class and method names to JSON.”

<user> RDoc, can you help me?
<rdoc> no.

<user> YARD, can you help me?

<yard> sure:

```
require 'yard'  
require 'json'
```

```
YARD::Registry.load  
objs = YARD::Registry.all(:class, :method).to_json
```

SRSLY?

srsly.

YARD:

Your Data is Safe With Me!*

* not an acronym

What Data?

I thought I was writing *docs*.

Oh right, you've been using *RDoc*.

WARNING: RDoc Comparison

- In RDoc, your comments are islands. Big blobs of text, no interconnection.
- YARD lets you formally declare parameters, return types, etc. in a standardized fashion.

Say it with me:

META DATA

Example

That's so meta.



```
# Sends a {Message} object out as an email using the configuration
# set during initialization.
#
# @param [Message] an email to send
# @raise [ArgumentError] if message is not a {Message} object
# @raise [TransportError] if message is not {Message#valid? valid}.
def mail(message)
  unless Message === message
    raise ArgumentError, "expected MmMail::Message, got #{message.class}"
  end

  raise TransportError, "invalid message" unless message.valid?

  send("mail_#{config.method}", message)
end
```

RDoc:

```
# Sends a Message object out as an email using the configuration
# set during initialization. Raises a TransportError if the message
# returns false for +#valid?+ and an ArgumentError if message is not
# a Message object.
def mail(message)
  unless Message === message
    raise ArgumentError, "expected MmMail::Message, got #{message.class}"
  end

  raise TransportError, "invalid message" unless message.valid?

  send("mail_#{config.method}", message)
end
```

Parameters, Return Types, Option Hashes, and more.

```
# Creates a new message with associated fields.-  
# -  
# @example-  
#   MmMail::Message.new(:to => 'test@example.com', :body => 'hi')-  
# -  
# @param [Hash] opts the options to create a message with.-  
# @option opts [String] :from ('nobody@localhost') The email's From field-  
# @option opts [String] :subject ('') The email's Subject field-  
# @option opts [String] :body ('') The email's body (not a header)-  
# @option opts [String] :to (nil) The email's To field. List multiple recipients as-  
#   'a@b.c, b@c.d', not an array.-  
def initialize(opts = {})-
```

Parameters:

Options Hash opts

Key Name	Default Value	Accepted Types	Description
:from	'nobody@localhost'	[String]	The email's From field
:subject	''	[String]	The email's Subject field
:body	''	[String]	The email's body (not a header)
:to	nil	[String]	The email's To field. List multiple recipients as 'a@b.c, b@c.d', not an array.

Declaring Types?

That's for suckers!

- a) They're optional
- b) Any useful API should be declaring types (even Ruby does it [poorly])
- c) You can declare more than just "types"

Like.....

Duck Types?

```
# @param [#read] file an object that responds to +file.read+  
def get_file_contents(file) ... end
```

```
*quack* *quack*  
      *quack*
```

This documentation thing...

It sounds like a lot of work.

It should be.

Simple Problem:

“Which methods in my API are currently deprecated?”

<user> RDoc, can you help me?

<rdoc> no.

<user> YARD, can you help me?

<yard> sure:

```
require 'yard'
```

```
YARD::Registry.load
```

```
YARD::Registry.all.select do |object|
```

```
  if object.has_tag?(:deprecated)
```

```
    puts object.path
```

```
  end
```

```
end
```

Extend Me.

YARD is built with extensibility as a *main* goal.

Why would I want to
extend a doc tool?

That sounds like a lot of work!*

* *It's not.*

a) Your API uses a Ruby DSL

document.this!

```
class MyClass  
  cattr_accessor :my_class_accessor  
  cattr_accessor :another_one  
end
```

YARD can get them in your docs in under 100 lines of code.

**b) You want to add
custom meta-data**

Or maybe just peek at existing meta-data

Force Declaration of Overriden Methods

```
class MyClass  
  # @override  
  def to_s; end  
  
  # yell at me  
  def object_id; end  
end
```

```
YARD::Tags::Library.define_tag 'Overriden Method', :override
```

Oh yea...

YARD can generate HTML docs too.

* queue demo *

UNIMPLEMENTED

Some Inspiring Ideas

Brought to you by Me.

rdoc.info

Bad name, awesome concept.

This one is (more-or-less) implemented :-)

Even better:

Fully dynamic doc server (a-la gem server)

- Searchable
 - Hosts user comments
 - Wiki-editable docs? Able to sync back to source?
-
- You can use YARD's marshal dump directly, or even even export it to an SQL database.

A better dcov

- dcov is unmaintained (last release was 2007).
- You could rewrite it with YARD in a few hundred lines.
- Lint your documentation:
 - Verify all your methods are documented (coverage ratio)
 - Make sure they declare the correct tags (@param, @return)
 - Make sure any @example emits the correct result

I heard DataMapper is doing this!

Testing Tools!

- Generate documentation for your tests
- Embed tests in your documentation
- and more...

YARD 0.2.3 (Milkshakes)



Bringing all the boys to the yard.

Website!

<http://yard.soen.ca>

IRC!

[#yard on irc.freenode.net](#)

Github!

<http://github.com/lsegal/yard>