DOCUMENTATION IS FREAKING AWESOME

a magical afternoon with Kyle Neath
Kyle Neath is...
~designer
@github
SOCIAL CODING
A favorite pastime...

Building small projects with ruby
There’s a library for almost everything
It’s not if the library exists...

It’s whether I can figure out how to use the @!&*# thing
Documentation is...
def extract_tags(data)
  data.gsub(/(.?)\[\[(.+?)\]\]\([^\]\]\[/m) do |$1, $2, $3|
    if $1 == '
      
      
    elsif $2.include?(']
      
      
    else
      id = Digest::SHA1.hexdigest($2)
      @tagmap[id] = $2
      
    end
end
end

# Process all tags from the tagmap and replace the placeholders with the
# final markup.
First there was RDoc
rdoc.sourceforge.net

“latest via CVS”
Welcome to Rails

Rails is a web-application framework that includes everything needed to create database-backed web applications according to the Model-View-Control pattern.

This pattern splits the view (also called the presentation) into “dumb” templates that are primarily responsible for inserting pre-built data in between HTML tags. The model contains the “smart” domain objects (such as Account, Product, Person, Post) that holds all the business logic and knows how to persist themselves to a database. The controller handles the incoming requests (such as Save New Account, Update Product, Show Post) by manipulating the model and directing data to the view.

In Rails, the model is handled by what’s called an object-relational mapping layer entitled Active Record. This layer allows you to present the data from database rows as objects and embellish these data objects with business logic methods. You can read more about Active Record in its README.

The controller and view are handled by the Action Pack, which handles both layers by its two parts: Action View and ActionController. These two layers are bundled in a single package due to their heavy interdependence. This is unlike the relationship between the Active Record and Action Pack that is much more separate. Each of these packages can be used independently outside of Rails. You can read more about Action Pack in its README.

Getting Started

1. Install Rails at the command prompt if you haven’t yet:
   ```
gem install rails
   ```

2. At the command prompt, create a new Rails application:
   ```
rails new myapp
   ```
   where “myapp” is the application name.

3. Change directory to myapp and start the web server:
Class **ArgumentError**

**In:** error.c

**Parent:** StandardError

**Ruby version:** Ruby 1.9.2

Raised when the arguments are wrong and there isn’t a more specific Exception class.

Ex: passing the wrong number of arguments

```
[1, 2, 3].first(4, 5)
```

*raises the exception:*

`ArgumentError: wrong number of arguments (2 for 1)`

Ex: passing an argument that is not acceptable:

```
[1, 2, 3].first(-4)
```

*raises the exception:*

`ArgumentError: negative array size`
Then there was YARD

yardoc.org
YARD: Yay! A Ruby Documentation Tool

Homepage: http://yardoc.org
IRC: irc.freenode.net / #yard
Git: http://github.com/lsegal/yard
Author: Loren Segal
Contributors: See Contributors section below
Copyright: 2007–2010
License: MIT License
Latest Version: 0.6.4 (codename "Snowy White Picket Fences")
Release Date: December 21st 2010

Synopsis

YARD is a documentation generation tool for the Ruby programming language. It enables the user to generate consistent, usable documentation that can be exported to a number of formats very easily, and also supports extending for custom Ruby constructs such as custom class level definitions. Below is a summary of some of YARD's notable features.

Feature List

1. RDoc/SimpleMarkup Formatting Compatibility: YARD is made to be compatible with RDoc formatting. In fact, YARD does no processing on RDoc documentation strings, and leaves this up to the output generation tool to decide how to render the documentation.

2. Yardoc Meta-tag Formatting Like Python, Java, Objective-C and other languages: YARD uses a '@tag' style definition syntax for meta tags alongside regular code documentation. These tags should be able to happily sit side by side RDoc formatted documentation, but provide a much more consistent and usable way to describe important information about objects, such as what parameters they take and what types they are expected to be, what type a method should return, what exceptions it can raise, if it is deprecated, etc.. It also allows information to be better (and more consistently) organized during the output generation phase. You can find a list of tags in the Tags.md file.

YARD also supports an optional "types" declarations for certain tags. This allows the developer to document type signatures for ruby methods and parameters in a non intrusive but helpful and consistent manner. Instead of describing this data in the body of the description, a developer may formally declare the parameter or return type(s) in a single line. Consider the following Yardoc'd method:
And of course TomDoc
tomdoc.org
FACT
TomDoc will save the world

Photo Credit: Claude Nix
Unless you want generated docs

Photo Credit: Claude Nix
YARD & RDoc are highly structured
# Reverses the contents of a String or IO object.
#
# @param [String, #read] contents the contents to reverse
# @return [String] the contents reversed lexically

def reverse(contents)
  ...
end
TomDoc is lightly structured
# Extract all code blocks into the codemap and replace
# with placeholders.
#
# data - The raw String data.
#
# Returns the placeholder'd String data.
def extract_code(data)
    ...
end
Also, tools like docco
rocco, pycco, shocco
Main Documentation Generation Functions

Generate the documentation for a source file by reading it in, splitting it up into comment/code sections, highlighting them for the appropriate language, and merging them into an HTML template.

Given a string of source code, parse out each comment and the code that follows it, and create an individual section for it. Sections take the form:

```javascript
{
  docs_text: ...
  docs_html: ...
  code_text: ...
  code_html: ...
}
```

Highlights a single chunk of CoffeeScript code, using Pygments over stdio, and runs the text of its corresponding comment through Markdown, using the Github-flavored-Markdown modification of Showdown.js.

We process the entire file in a single call to Pygments by inserting little marker comments between each section and then splitting the result string wherever our markers occur.

```javascript
generate_documentation = (source, callback) ->
  fs.readFile source, "utf-8", (error, code) ->
    throw error if error
    sections = parse source, code
    highlight source, sections, ->
      generate_html source, sections
    callback()

parse = (source, code) ->
  lines = code.split '\n'
  sections = []
  language = get_language source
  has_code = docs_text = code_text = ''

  save = (docs, code) ->
    sections.push docs: docs, code: code

  for line in lines
    if line.match language.comment_matcher
      if not (line.match language.comment_filter)
        if has_code
          save docs_text, code_text
          has_code = docs_text = code_text = ''
          docs_text += line.replace(language.comment_matcher, '') + '\n'
        else
          has_code = yes
          code_text += line + '\n'
    save docs_text, code_text
    sections

highlight = (source, sections, callback) ->
  language = get_language source
  pygments = spawn 'pygmentize', ['-l', language.name, '-f', 'html', '-o', '0', 'encoding=utf-8']
  output = ''
  pygments.stderr.addListener 'data', (error) ->
    console.error error if error
  pygments.stdout.addListener 'data', (result) ->
    output += result if result
  pygments.addListener 'exit', () ->
    output = output.replace(highlight_start, '').replace(highlight_end, '')
    fragments = output.split language.divider_html
    for section, i in sections
      section.code_html = highlight_start + fragments[i] + highlight_end
      section.docs_html = showdown.makeHtml section.docs_text
    callback()

  pygments.stdin.write((section.code_text for section in sections).join(language.divider_text))
  pygments.stdin.end()
```
Main Documentation Generation Functions

Generate the documentation for a source file by reading it in, splitting it up into comment/code sections, highlighting them for the appropriate language, and merging them into an HTML template.

Given a string of source code, parse out each comment and the code that follows it, and create an individual section for it. Sections take the form:

```javascript
{
  docs_text: ...
  docs_html: ...
  code_text: ...
  code_html: ...
}
```
generate_documentation = (source, callback) ->
    fs.readFile source, "utf-8", (error, code) ->
      throw error if error
      sections = parse source, code
      highlight source, sections, ->
        generate_html source, sections
        callback()

parse = (source, code) ->
  lines    = code.split "\n"
  sections = []
  language = get_language source
  has_code = docs_text = code_text = ''

save = (docs, code) ->
  sections.push docs_text: docs, code_text: code

for line in lines
  if line.match language.comment_matcher
    if not (line.match language.comment_filter)
      if has_code
        save docs_text, code_text
        has_code = docs_text = code_text = ''
      docs_text += line.replace(language.comment_matcher, '') + 'n'
Code comments are just the start
AN AWESOME WEBSITE
Does your project Google?
Bundler: The best way to manage Ruby applications

The best way to manage your application's dependencies

Bundler manages an application's dependencies through its entire life across many machines systematically and repeatably.

I am interested in

- Reporting a Bug
- Understanding Bundler
- Gemfile Manual
- CLI Manual

Getting Started

Getting started with bundler is easy

```
$ gem install bundler
```

If you're on an old version of Rubygems (before 1.3.6)
What it does

Bundler manages an **application's dependencies** through its entire life across many machines systematically and repeatably.

I am interested in

- Reporting a Bug
- Understanding Bundler
- Gemfile Manual
- CLI Manual

Getting Started

Getting started with bundler is easy

```bash
$ gem install bundler
```
Bundler: The best way to manage Ruby applications

What it does

Bundler manages an application's dependencies through its entire life across many machines systematically and repeatably.

Getting Started

Getting started with bundler is easy

$ gem install bundler

If you're on an old version of Rubygems (before 1.3.6)
Bundler: The best way to manage your application's dependencies.

Bundler manages an application's dependencies through its entire life across many machines systematically and repeatably.

I am interested in

- Reporting a Bug
- Understanding Bundler
- Gemfile Manual
- CLI Manual

Getting Started

Getting started with bundler is easy

$ gem install bundler

If you're on an old version of Rubygems (before 1.3.6)
Bundler: The best way to manage Ruby applications

What it does
Bundler manages an application's dependencies through its entire life across many machines, automatically and repeatably.

How to Contribute

Bug Reports
I am interested in
- Reporting a Bug
- Understanding Bundler
- Gemfile Manual
- CLI Manual

Getting Started
Getting started with bundler is easy
$ gem install bundler

If you're on an old version of Rubygems (before 1.3.6)
(plus, it’s pretty)
Great place to post long form tutorials
Ruby on Rails Guides (v3.0.3)

These are the new guides for Rails 3. The guides for Rails 2.3 are still available at http://guides.rubyonrails.org/v2.3.8/.

These guides are designed to make you immediately productive with Rails, and to help you understand how all of the pieces fit together.

Start Here

Getting Started with Rails
Everything you need to know to install Rails and create your first application.

Models

Rails Database Migrations
This guide covers how you can use Active Record migrations to alter your database in a structured and organized manner.

Active Record Validations and Callbacks
This guide covers how you can use Active Record validations and callbacks.

Active Record Associations
This guide covers all the associations provided by Active Record.
AN AWESOME README
First Contact
Elements of a great README
Think about writing your README first
(readme driven development)
LOTS AND LOTS OF MAN PAGES
WTF is a man page?
Documentation for UNIX tools
(command line programs)
NAME
rails -- Web-application framework

SYNOPSIS
rails path [options...]

DESCRIPTION
Rails is a web-application and persistence framework that includes everything needed to create database-backed web-applications according to the Model-View-Control pattern of separation. This pattern splits the view (also called the presentation) into "dumb" templates that are primarily responsible for inserting pre-built data in between HTML tags. The model contains the "smart" domain objects (such as Account, Product, Person, Post) that holds all the business logic and knows how to persist themselves to a database. The controller handles the incoming requests (such as Save New Account, Update Product, Show Post) by manipulating the model and directing data to the view.

Rails is written with the ruby(1) language. For more information about Rails you can use its --help flag. There is also online documentation available at "http://rubyonrails.org".

SEE ALSO
ruby(1) mongrel_rails(1) cap(1)

AUTHORS
Rails was created by David Heinemeier Hansson <david@loudthinking.com> then extended and improved by hundreds of open-source contributors.
Many sections

~$ man 5 mustache
mustache(5) - Mustache Syntax
mustache(1) - Usage of `mustache`
BIG CAVEAT

gems don’t install man pages :

check out gem-man until then
Documentation is...
CODE COMMENTS

Available with the source

Great for Contributors
AWESOME WEBSITE

Google Juice

Command center for docs
AWESOME README

Available with the source
First contact with docs
LOTS OF MAN PAGES

Available in terminal

First place UNIX nerds look
Documentation is...

A Great Marketing Tool
First contact with your project

(make it count!)
More Docs = Better Perceived Quality
More Docs
=
Easier To Learn
More Docs = Easier To Contribute
tldr;
More People Using Your Project
“Top ten reasons why I won't use your open source project”

Wynn Netherland
pengwynn
REASON #1
You don’t have a friggin’ Readme
REASON #3
You have no project home page
Documentation is important marketing
Documentation is...

Therapeutic
Forces you to slow down
Puts you into a different mindset
Forces you to question your code
Explaining code often reveals flaws

(like an invisible pairing partner)
It can also be a great stress reliever

sometimes code just sucks
Knowing how someone feels about code is valuable.

# XXX: I hate myself and want to die.
# --rtomayko 2010-05-27
At the end of the day...

Writing documentation produces higher quality code
rdoc.info

Automatic YARD Generation
rdoc.info

GitHub Integration
(generate docs on push)
Welcome to \Rails

\Rails is a web-application framework that includes everything needed to create database-backed web applications according to the Model-View-Control pattern.

This pattern splits the view (also called the presentation) into "dumb" templates that are primarily responsible for inserting pre-built data in between HTML tags. The model contains the "smart" domain objects (such as Account, Product, Person, Post) that hold all the business logic and knows how to persist themselves to a database. The controller handles the incoming requests (such as Save New Account, Update Product, Show Post) by manipulating the model and directing data to the view.

In \Rails, the model is handled by what's called an object-relational mapping layer entitled Active Record. This layer allows you to present the data from database rows as objects and embellish these data objects with business logic methods. You can read more about Active Record in its README.

The controller and view are handled by the Action Pack, which handles both layers by its two parts: Action View and Action Controller. These two layers are bundled in a single package due to their heavy interdependence. This is unlike the relationship between the Active Record and Action Pack that is much more separate. Each of these packages can be used independently outside of \Rails. You can read more about Action Pack in its README.

Getting Started

1. Install \Rails at the command prompt if you haven't yet:
   ```bash
   gem install rails
   ```

2. At the command prompt, create a new \Rails application:
   ```bash
   rails new myapp
   ```
   where "myapp" is the application name.

3. Change directory to myapp and start the web server:
   ```bash
   cd myapp; rails server
   ```
   Run with --help for options.

4. Go to localhost:3000/ and you'll see:
   ```bash
   "Welcome aboard: You're riding Ruby on Rails!"
   ```

5. Follow the guidelines to start developing your application. You can find the following resources handy:
gem server

Locally Generated RDoc

```
$ gem server
Server started at http://0.0.0.0:8080
Server started at http://[::ffff:0.0.0.0]:8080
```

**Summary**

There are 253 gems installed:

- abstract
- actionmailer
- actionmailer
- actionmailer
- actionmailer
- actionmailer
- actionmailer
- actionpack
- actionpack
- actionpack
- actionpack
- actionpack
- actionpack
- actionpack
- actionpack
RailsAPI.com

Awesome find-as-you-type Ruby/Rails/Gem Docs
Welcome to Rails

Rails is a web-application framework that includes everything needed to create database-backed web applications according to the Model-View-Control pattern.

This pattern splits the view (also called the presentation) into “dumb” templates that are primarily responsible for inserting pre-built data in between HTML tags. The model contains the “smart” domain objects (such as Account, Product, Person, Post) that holds all the business logic and knows how to persist themselves to a database. The controller handles the incoming requests (such as Save New Account, Update Product, Show Post) by manipulating the model and directing data to the view.

In Rails, the model is handled by what’s called an object-relational mapping layer entitled Active Record. This layer allows you to present the data from database rows as objects and embellish these data objects with business logic methods. You can read more about Active Record in files/vendor/rails/active_record/README.html.

The controller and view are handled by the Action Pack, which handles both layers by its two parts: Action View and Action Controller. These two layers are bundled in a single package due to their heavy interdependence. This is unlike the relationship between the Active Record and Action Pack that is much more separate. Each of these packages can be used independently outside of Rails. You can read more about Action Pack in files/vendor/rails/actionpack/README.html.

Getting Started

1. At the command prompt, start a new Rails application using the rails command and your application name. Ex: rails myapp
2. Change directory into myapp and start the web server: script/server (run with --help for options)
3. Go to localhost:3000/ and get “Welcome aboard: You’re riding the Rails!”
4. Follow the guidelines to start developing your application

Web Servers

By default, Rails will try to use Mongrel and lighttpd if they are installed, otherwise Rails will use WEBrick, the webserver that ships with Ruby. When you run script/server, Rails will check if Mongrel exists, then lighttpd and finally fall back to WEBrick. This ensures that you can always get up and running quickly.
railsapi
+
Fluid.app

Awesome offline Ruby & Rails documentation
jqapi.com

Like railsapi.com, but for jQuery
Write man pages in markdown

[github.com/rtomayko/ronn](https://github.com/rtomayko/ronn)
man pages are written in roff

```
\fBronn\fR < \fIfile\fR
.
.SH "DESCRIPTION"
\fBRonn\fR converts textfiles to standard roff- formatted UNIX manpages or HTML\. ronn\-format(7) is based on markdown(7) but includes additional rules and syntax geared toward authoring manuals\.
```

""
**DESCRIPTION**

`ronn` converts textfiles to standard roff- or html-formatted UNIX manpages. `ronn` is based on markdown, but includes additional rules and syntax.
use ronn instead

## DESCRIPTION

**Ronn** converts textfiles to standard roff-formatted UNIX manpages or HTML. **ronn-format**(7) is based on **markdown**(7) but includes additional rules and syntax geared toward authoring manuals.

Get HTML generation for free
NAME
mustache -- Logic-less templates.

SYNOPSIS
A typical Mustache template:

Hello {{name}}
You have just won ${{value}}!
{{#in_ca}}
Well, ${{taxed_value}}, after taxes.
{{/in_ca}}

Given the following hash:

{ 
  "name": "Chris",
  "value": 10000,
  "taxed_value": 10000 - (10000 * 0.4),
  "in_ca": true
}

Will produce the following:

Hello Chris
You have just won $10000!
Well, $6000.0, after taxes.

DESCRIPTION
Mustache can be used for HTML, config files, source code - anything. It
works by expanding tags in a template using values provided in a hash or
object.

We call it "logic-less" because there are no if statements, else clauses,
or for loops. Instead there are only tags. Some tags are replaced with a
value, some nothing, and others a series of values. This document
Final Thoughts
Documentation is a lot more than RDoc
Documenting should be something you **want to do**

**ProTip:** It’s not a guilt trip
Documenting is a great marketing tool
Documenting helps you write better code
... and always keep an offline version of your docs
Fin.

warpspire.com/talks/documentation