== 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

link:files/vendor/rails/activerecord/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

link:files/vendor/rails/actionpack/README.html.

== Getting Started

1. At the command prompt, create a new Rails application:

<tt>rails new myapp</tt> (where <tt>myapp</tt> is the application name)

2. Change directory to <tt>myapp</tt> and start the web server:

<tt>cd myapp; rails server</tt> (run with --help for options)

3. Go to http://localhost:3000/ and you'll see:

"Welcome aboard: You're riding Ruby on Rails!"

4. Follow the guidelines to start developing your application. You can find

the following resources handy:

\* The Getting Started Guide: http://guides.rubyonrails.org/getting\_started.html

\* Ruby on Rails Tutorial Book: http://www.railstutorial.org/

== Debugging Rails

Sometimes your application goes wrong. Fortunately there are a lot of tools that

will help you debug it and get it back on the rails.

First area to check is the application log files. Have "tail -f" commands

running on the server.log and development.log. Rails will automatically display

debugging and runtime information to these files. Debugging info will also be

shown in the browser on requests from 127.0.0.1.

You can also log your own messages directly into the log file from your code

using the Ruby logger class from inside your controllers. Example:

class WeblogController < ActionController::Base

def destroy

@weblog = Weblog.find(params[:id])

@weblog.destroy

logger.info("#{Time.now} Destroyed Weblog ID ##{@weblog.id}!")

end

end

The result will be a message in your log file along the lines of:

Mon Oct 08 14:22:29 +1000 2007 Destroyed Weblog ID #1!

More information on how to use the logger is at http://www.ruby-doc.org/core/

Also, Ruby documentation can be found at http://www.ruby-lang.org/. There are

several books available online as well:

\* Programming Ruby: http://www.ruby-doc.org/docs/ProgrammingRuby/ (Pickaxe)

\* Learn to Program: http://pine.fm/LearnToProgram/ (a beginners guide)

These two books will bring you up to speed on the Ruby language and also on

programming in general.

== Debugger

Debugger support is available through the debugger command when you start your

Mongrel or WEBrick server with --debugger. This means that you can break out of

execution at any point in the code, investigate and change the model, and then,

resume execution! You need to install ruby-debug to run the server in debugging

mode. With gems, use <tt>sudo gem install ruby-debug</tt>. Example:

class WeblogController < ActionController::Base

def index

@posts = Post.all

debugger

end

end

So the controller will accept the action, run the first line, then present you

with a IRB prompt in the server window. Here you can do things like:

>> @posts.inspect

=> "[#<Post:0x14a6be8

@attributes={"title"=>nil, "body"=>nil, "id"=>"1"}>,

#<Post:0x14a6620

@attributes={"title"=>"Rails", "body"=>"Only ten..", "id"=>"2"}>]"

>> @posts.first.title = "hello from a debugger"

=> "hello from a debugger"

...and even better, you can examine how your runtime objects actually work:

>> f = @posts.first

=> #<Post:0x13630c4 @attributes={"title"=>nil, "body"=>nil, "id"=>"1"}>

>> f.

Display all 152 possibilities? (y or n)

Finally, when you're ready to resume execution, you can enter "cont".

== Console

The console is a Ruby shell, which allows you to interact with your

application's domain model. Here you'll have all parts of the application

configured, just like it is when the application is running. You can inspect

domain models, change values, and save to the database. Starting the script

without arguments will launch it in the development environment.

To start the console, run <tt>rails console</tt> from the application

directory.

Options:

\* Passing the <tt>-s, --sandbox</tt> argument will rollback any modifications

made to the database.

\* Passing an environment name as an argument will load the corresponding

environment. Example: <tt>rails console production</tt>.

To reload your controllers and models after launching the console run

<tt>reload!</tt>

More information about irb can be found at:

link:http://www.rubycentral.org/pickaxe/irb.html

== dbconsole

You can go to the command line of your database directly through <tt>rails

dbconsole</tt>. You would be connected to the database with the credentials

defined in database.yml. Starting the script without arguments will connect you

to the development database. Passing an argument will connect you to a different

database, like <tt>rails dbconsole production</tt>. Currently works for MySQL,

PostgreSQL and SQLite 3.

== Description of Contents

The default directory structure of a generated Ruby on Rails application:

|-- app

| |-- assets

| |-- images

| |-- javascripts

| `-- stylesheets

| |-- controllers

| |-- helpers

| |-- mailers

| |-- models

| `-- views

| `-- layouts

|-- config

| |-- environments

| |-- initializers

| `-- locales

|-- db

|-- doc

|-- lib

| `-- tasks

|-- log

|-- public

|-- script

|-- test

| |-- fixtures

| |-- functional

| |-- integration

| |-- performance

| `-- unit

|-- tmp

| |-- cache

| |-- pids

| |-- sessions

| `-- sockets

`-- vendor

|-- assets

`-- stylesheets

`-- plugins

app

Holds all the code that's specific to this particular application.

app/assets

Contains subdirectories for images, stylesheets, and JavaScript files.

app/controllers

Holds controllers that should be named like weblogs\_controller.rb for

automated URL mapping. All controllers should descend from

ApplicationController which itself descends from ActionController::Base.

app/models

Holds models that should be named like post.rb. Models descend from

ActiveRecord::Base by default.

app/views

Holds the template files for the view that should be named like

weblogs/index.html.erb for the WeblogsController#index action. All views use

eRuby syntax by default.

app/views/layouts

Holds the template files for layouts to be used with views. This models the

common header/footer method of wrapping views. In your views, define a layout

using the <tt>layout :default</tt> and create a file named default.html.erb.

Inside default.html.erb, call <% yield %> to render the view using this

layout.

app/helpers

Holds view helpers that should be named like weblogs\_helper.rb. These are

generated for you automatically when using generators for controllers.

Helpers can be used to wrap functionality for your views into methods.

config

Configuration files for the Rails environment, the routing map, the database,

and other dependencies.

db

Contains the database schema in schema.rb. db/migrate contains all the

sequence of Migrations for your schema.

doc

This directory is where your application documentation will be stored when

generated using <tt>rake doc:app</tt>

lib

Application specific libraries. Basically, any kind of custom code that

doesn't belong under controllers, models, or helpers. This directory is in

the load path.

public

The directory available for the web server. Also contains the dispatchers and the

default HTML files. This should be set as the DOCUMENT\_ROOT of your web

server.

script

Helper scripts for automation and generation.

test

Unit and functional tests along with fixtures. When using the rails generate

command, template test files will be generated for you and placed in this

directory.

vendor

External libraries that the application depends on. Also includes the plugins

subdirectory. If the app has frozen rails, those gems also go here, under

vendor/rails/. This directory is in the load path.