Rails Quick Reference

January 7, 2007

Abstract

Ruby on Rails is a large framework with many of built-in methods. This list covers many of the features we will discuss and use during the workshop. The complete documentation for Ruby on Rails is available at http://edgedocs.planetargon.org/

Contents

1 General

1.1 Documentation

1.1.1 Official Rail API

- http://api.rubyonrails.org

1.1.2 Searchable Rails API

- http://rails.outertrack.com
- http://railshelp.com

1.1.3 Ruby Documentation

- http://ruby-doc.org

1.2 Supported Web Servers

- WEBrick
- Mongrel
- Lighttpd
- Apache
- MS IIS

1.3 Supported Databases

- DB2
- Firebird
- MySQL
- Oracle
- PostreSQL
- SQLite
- SQL Server

1.4 Integrated Development Environments

1.4.1 Open Source

- Eclipse/RDT *http://rubyeclipse.sourceforge.net*
- FreeRIDE *http://freeride.rubyforge.org*
- RadRails (built with Eclipse/RDT) *http://www.radrails.org*
- RDE (Ruby Development Environment) *http://homepage2.nifty.com/sakazuki/rde_e.html*

1.4.2 Commercial

- ArachnoRuby *http://www.ruby-ide.com/ruby/ruby_ide_and_ruby_editor.php*
- Komodo *http://www.activestate.com/Products/Komodo*

1.5 Create a New Rails Application

```
rails app_name
```

Options:

```
-d=xxx or --database=xxx (specify database to use; defaults to mysql)
-r=xxx or --ruby-path=xxx (specify the path to Ruby)
-f or --freeze (Freezes rails into the vendor directory)
```

2 Active Record

The ActiveRecord class is at the heart of Rails. It is a powerful way to map database tables to objects within the application.

2.1 Automated Mapping

Automatically maps:

- Tables —> classes
- Rows —> objects (instances of model classes)
- Columns —> object attributes

Table to class mapping uses English plurals:

- An `Invoice` model class maps to an `invoices` table.
- A `Person` model class maps to a `people` table.
- A `Country` model class maps to a `countries` table.
- A `SecurityLevel` model class maps to a `security_levels` table.

2.2 Associations

Four ways of associating models:

1. has_one
2. has_many
3. belongs_to
4. has_and_belongs_to_many

Examples

```
def Order < ActiveRecord::Base
  has_many :line_items
  belongs_to :customer # there's a column 'customer_id' in the orders table
end

def LineItem < ActiveRecord::Base
  has_many :orders
  has_one :address
end

def Address < ActiveRecord::Base
  belongs_to :customer
end
```

2.3 Validations

```
validates_presence_of :firstname, :lastname    # must be filled out

validates_length_of :password,
                    :minimum => 8,                # more than 8 characters
                    :maximum => 16,               # shorter than 16 characters
                    :in => 8..16,                 # between 8 and 16 characters
                    :too_short => 'way too short',
                    :too_long  => 'way to long'

validates_acceptance_of :eula                     # must accept a condition
                        :accept => 'Y'            # default: 1 (ideal for checkbox)

validates_format_of :email
                    :with => /^(.+)@((?:[-a-z0-9]+\.)[a-z]{2,})$/i

validates_numericality_of :value,                 # value is numeric
                          :only_integer => true
                          :allow_nil => true

validates_inclusion_in :gender,       # a value in an enumeration (enum) field
                       :in => %w( m, f )

validates_exclusions_of :age          # value not in enumeration (enum) field
                        :in => 13..19 # don't want teenagers
```

2.3.1 Validation options

```
:message => 'a personalized message'
:on      => :create                   # or :update (validates only then)
:if      => ...                       # call a method or Proc
```

2.4 Calculations

```
Person.average :age
Person.minimum :age
Person.maximum :age
Person.count
```

```
Person.count(:conditions => "age > 26"
Person.sum :salary, :group => :last_name
```

2.5 Finders

```
find(43)
find([37, 42])
find :all
find :first, :conditions => [ "name = ?", 'Julia Roberts']
```

2.5.1 More parameters for `find`

```
:order => 'name DESC'                                  # SQL fragment
:offset => 20                                          # starts with entry 20
:limit => 10                                           # only return 10 objects
:group => 'name'                                       # SQL fragment GROUP BY
:joins => 'LEFT JOIN ...'                              # additional LEFT JOIN (rarely used)
:include => [:account, :friends]                       # LEFT OUTER JOIN with these models
:include => {:groups => {:members => {:favorites }}}
:select => [:name, :address]                           # instead of SELECT * FROM
:readonly => true                                      # objects are write protected
```

2.5.2 Dynamic Attribute-based Finders

```
Person.find_by_user_name(user_name)
Person.find_all_by_last_name(last_name)
Person.find_by_user_name_and_password(user_name,password)
Order.find_by_name("Julia Roberts")
```

2.6 Migrations

```
% ruby script/generate migration MyAddTables
```

creates `db/migrations/001_my_add_tables.rb`

```
class MyAddTables < ActiveRecord::Migration
  def self.up
  end
  def self.down
  end
end
```

The methods `up()` and `down()` change the db schema:

```
  def self.up   # brings db schema to the next version
    create_table :table do |t|
      t.column :name, :string
      t.column :age, :integer, { :default => 42 }
      t.column :description, :text
      # :string, :text, :integer, :float, :datetime, :timestamp, :time, :date,
      # :binary, :boolean
  end

  add_column :table, :column, :type
  rename_column :table, :old_name, :new_name
```

```
    change_column :table, :column, :new_type
    execute "SQL statement"
    add_index :table, :column, :unique => true, :name => 'some_name'
    add_index :table, [:column1, :column2]
    def self.down  #rollback changes
      rename_column :table, :new_name, :old_name
      remove_column :table, :column
      drop_table :table
      remove_index :table, :column
    end
  end
```

To execute the migration:

```
% rake db:migrate
% rake db:migrate VERSION=14
% rake db:migrate RAILS_ENV=production
```

3 Controllers

3.1 Controller Methods

Each public method in a controller is callable in default URL scheme `/controller/action` (`/hello/world` in the example):

```
class WorldController < ApplicationController
  def hello
    render :text => 'Hello, world'
  end
end
```

All request parameters, both from a GET or POST request, or from the URL, are available through the `params` hash:

```
/world/hello/1?foo=bar
id = params[:id]
foo = params[:bar]
```

Determine the type of response accepted:

```
def index
  @posts = Post.find :all
  respond_to do |type|
    type.html  # using default, which will render weblog/index.html
    type.xml  { render :action => "index.rxml" }
    type.js   { render :action => "index.rjs" }
  end
end
```

3.2 Render

Usually the view template with the same name as the controller method is used to render results.

3.2.1 Action

```
render :action => 'some_action'    # the default. Does not need to be specified
                                   # in a controller method called 'some_action'
render :action => 'another_action', :layout => false
render :action => 'some_action', :layout => 'another_layout'
```

3.2.2 Partials

Partials are stored in files whose filenames begin with an underscore (like `_post`, `_form`, and `_item`):

```
render :partial => 'post'
render :partial => 'error', :status => 500
render :partial => 'form', :locals => { :variable => @another_variable }
render :partial => 'item', :collection => @list
render :partial => 'item, :collection => @list, :spacer_template => 'list_divider'
```

3.2.3 Templates

Similar to rendering an action, but finds the template based on the template root (app/views):

```
render :template => 'weblog/show'  # renders app/views/weblog/show
```

3.2.4 Files

```
render :file => '/path/to/some/file'
render :file => '/path/to/some/filenotfound.rhtml', :status => 404, :layout => true
```

3.2.5 Text

```
render :text => 'Hello World!'
render :text => "This is an error", :status => 500
render :text => "Let's us a layout", :layout => true
render :text => "Specific layout", :layout => 'special'
```

3.2.6 Inline Template

Uses ERb to render the "miniature" template:

```
render :inline => "<%= 'hello, ' * 3 + 'again' %>"
render :inline => "<%= 'hello ' + name %>", :locals => { :name => 'David' }
```

3.2.7 RJS

Javascript templates:

```
def refresh
  render :update do |page|
    page.replace_html 'user_list', :partial => 'user', :collection => @users
    page.visual_effect :highlight, 'user_list'
  end
end
```

3.2.8 Change content_type

```
render :action => 'atom.xml', :content_type => 'application/atom+xml'
```

3.2.9 Redirects

```
redirect_to :action => 'edit'
redirect_to :controller => 'accounts', :action => 'signup'
```

3.2.10 Nothing

```
render :nothing
render :nothing, :status => 403   # forbidden
```

3.3 URL Routing

In *config/routes.rb*

```
map.connect '', :controller => 'posts', :action => 'list'  # default
map.connect ':action/:controller/:id'
map.connect 'tasks/:year/:month', :controller => 'tasks',
                                  :action => 'by_date'
                                  :month => nil, :year => nil,
                                  :requirements => { :year => /\d{4}/,
                                                     :month => /\d{1,2}/ }
```

3.4 Filter

Filters can change a request before or after the controller. They can, for example, be use for authentication, encryption, or compression:

```
before_filter :login_required, :except => [ :login ]
before_filter :authenticate, :only => [ :edit, :delete ]
after_filter :compress
```

It's also possible to use a `proc` for a really small filter action:

```
before_filter { |controller| false if controller.params["stop_action"] }
```

3.5 Session/Flash

To save data across multiple requests, you can use either the session or the flash hashes. A flash stores a value (normally text) until the next request, while a session stores data during the complete session:

```
session[:user] = @user
flash[:message] = 'Data was saved successfully'

<%= link_to 'login', :action => 'login' unless session[:user] %>
<% if flash[:message] %>
<div><%= h flash[:message %></div>
<% end %>
```

3.6 Session Management

It's possible to turn off session management:

```
session :off                      # turn session management off
session :off, :only => :action    # only for this :action
session :off, :except => :action  # except for this :action
session :only => :foo,            # only for :foo when doing HTTPS
        :session_secure => true
session :off, :only => :foo,      # off for :foo, if uses a Web Service
        :if => Proc.new { |req| req.parameters[:ws] }
```

3.7 Cookies

3.7.1 Setting

```
cookies[:user_name] = 'angelina jolie'   #  sets a simple session cookie
cookies[:login] = { :value => "XJ=122", :expires => Time.now + 3600 }
     # sets cookie that will expire in 1 hour
```

3.7.2 Reading

```
cookies[:user_name]   # => 'angelina jolie'
cookies.size          # => 2
```

3.7.3 Deleting

```
cookies.delete :user_name
```

3.7.4 Optional Symbols for Setting Cookies

value The cookie's value or list of values (as an array)

path The path for which this cookie applies (defaults to the root of the applicaton)

domain The domain for which this cookie applies

expires The time at which this cookie expires, as a Time object

secure Whether this cookie is a secure cookie (default to false). Secure cookies are transmitted only to HTTPS servers.

4 Views

View templates are stored in *app/views/controllername*. The extension determines the type of the template:

`*.rhtml` Ruby HTML (using ERb)

`*.rxml` Ruby XML (using Builder)

`*.rjs` Ruby Javascript

All instance variables (@variables) of the controller are available to the view. In addition, the following special objects can be accessed:

headers The headers of the outgoing response

request The incoming request object

response The outgoing response object

params The parameter hash

session The session hash

controller The current controller

4.1 RHTML

RHTML is HTML mixed with Ruby, using tags. All of Ruby is available for programming.

```
<% %>   # executes Ruby code
<%= %>  # executes Ruby code and displays the result

<ul>
<% @products.each do |p| %>
  <li><%= p.name %></li>
<% end %>
</ul>
```

The output of anything in <%= %> tags is directly copied to the HTML output stream. To secure against HTML injection, use the h() function to HTML-escape the output. For example:

```
<%=h @user_entered_notes %>
```

4.2 RXML

Creates XML files:

```
xml.instruct!                                  # <?xml version="1.0" encoding="UTF-8"?>
xml.comment!                                   # <!-- a comment -->
xml.feed "xmins" => "http://www.w3.org/2005/Atom" do
  xml.title "My Atom Feed"
  xml.subtitle h(@feed.subtitle), "type" => 'html'
  xml.link url_for(  :only_path => false,
                     :controller => 'feed',
                     :action => 'atom' )
  xml.updated @updated.iso8601
  xml.author do
    xml.name "Kevin Federline"
    xml.email "kfed01@ccsf.edu"
  end
  @entries.each do |entry|
    xml.entry do
      xml.title entry.title
      xml.link "href" => url_for ( :only_path => false,
                                   :controller => 'entries'
                                   :action => 'show',
                                   :id => entry )
      xml.id entry.urn
      xml.updated entry.updated.iso8601
      xml.summary h(entry.summary)
    end
  end
end
```

4.3 Helpers

Small functions, normally used for displaying data, can be extracted to helpers. Each view has its own helper class (in *app/helpers*). Common functionality is stored in *app/helpers/application_helper.rb*.

4.3.1 Links

```
link_to "Name", :controller => 'post, :action => 'show', :id => @post.id
link_to 'Delete',  { :controller => 'admin',
                     :action => 'delete',
                     :id => @post.id },
                   { :class => 'css-class',
                     :id => 'css-id',
                     :confirm => 'Are you sure?' }
# generates this HTML
<a href="/admin/delete/3" class="css-class" id="css-id"
    onclick="return confirm('Are you sure?');">Delete</a>

image_tag 'spinner.png', :class => 'image', :alt => 'Spinner'

mail_to 'info@hollywoodhotline.com', 'send mail',
    :subject => "Support request from #{@user.name}",
    :cc =>  @user.email
    :bcc => 'security@hollywoodhotline.com',
    :body => '....',
    :encoding => 'javascript'

stylesheet_link_tag 'cs132x', 'admin', :media => 'all'
```

4.4 HTML Forms

4.4.1 Form

```
<%= form_tag :action => 'save', :id => @product do, {:method => 'POST'} do %>
...
<% end %>
```

Use `:multipart => true` to define a MIME-multipart form for file uploads.

4.4.2 Text Fields

```
<%= text_field :modelname, :attribute_name, options %>
```

Creates this HTML

```
<input type="text" name="modelname[attribute_name]" id="attribute_name" />
```

Example:

```
<%= text_field "post", "title", :size => 20 %>
```

Creates this HTML

```
<input type="text" id="post_title" name="post[title]" size="20" value="#{@post.title}" />
```

Create a hidden field:

```
<%= hidden_field ... %>
```

Create a password field (input show as stars)

```
<%= password_field ... %>
```

Create a file field:

```
<%= file_field ... %>
```

4.4.3 Text Area

<%= text_area ... %>
This example:

```
<%= text_area ”post”, ”body”, :cols => 20, :rows => 40 %>
```

generates:

```
<textarea cols="20" id="post_body" name="post[body]" rows="40">#{@post.body}</textarea>
```

4.4.4 Radio Button

```
<%= radio_button :modelname, :attribute, :tag_value, options %>
```

Example:

```
<%= radio_button ”post”, ”category”, ”rails”
<%= radio_button ”post”, ”category”, ”java”
```

generates:

```
<input type=”radio” id=”post_category” name=”post[category]” value=”rails”
checked=”checked” />
<input type=”radio” id=”post_category” name=”post[category]” value=”java” />
```

4.4.5 Checkbox

```
<%= check_box :modelname, :attribute, options, on_value, off_value %>
```

Example:

```
<%= check_box ”post”, ”validate” %>   # post.validated? returns 1 or 0
```

generates:

```
<input id="post_validate" name="post[validate]" type="checkbox" value="1" />
<input name="post[validate]" type="hidden" value="0" />
```

4.4.6 Options

Creates a *select* tag. Pass an array of choices:

```
<%= select :variable, :attribute, choices, options, html_options %>
```

Example:

```
<%= select :post,
           :title,
           Post.find_all.collect { |p| [ p.title,p.id ] },
           { :include_blank => true}
%>
```

generates:

```
<select id="post_title" name="post[title]"><option value=""></option>
<option value="1">This is the first POST!</option>
<option value="2">better fill is it in</option>
<option value="3">City College of San Francisco is Cool</option></select>
```

4.4.7 Date and Time

```
<%= date_select :variable, :attribute, options %>
<%= datetime_select :variable, :attribute, options %>
```

Examples:

```
<%= date_select 'post', 'created_at' %>
<%= date_select 'user', 'birthday', :start_year => 1910 %>
<%= date_select 'user', "cc_date', :start_year => 2005,
                                   :use_month_numbers => true,
                                   :discard_day => true,
                                   :order => [:year,:month ] %>
<%= datetime_select 'post', 'created_at', :start_year => 2005, :discard_day => true %>
```

generates:

```
<form>
<select id="post_created_at_1i" name="post[created_at(1i)]">
<option value="2005">2005</option>
<option value="2006">2006</option>
<option value="2007" selected="selected">2007</option>
<option value="2008">2008</option>
<option value="2009">2009</option>
<option value="2010">2010</option>
<option value="2011">2011</option>
<option value="2012">2012</option>
</select>
<select id="post_created_at_2i" name="post[created_at(2i)]">
<option value="1" selected="selected">January</option>
<option value="2">February</option>
<option value="3">March</option>
<option value="4">April</option>
<option value="5">May</option>
<option value="6">June</option>
<option value="7">July</option>
<option value="8">August</option>
<option value="9">September</option>
<option value="10">October</option>
<option value="11">November</option>
<option value="12">December</option>
</select>
</form>
```

5 Layouts

A layout defines the surroundings of an HTML page. You can use layouts to define a common look and feel. Layouts reside in *app/views/layouts*.

Example layout:

```
<html>
  <head>
    <title>Form: <%= controller.action_name %></title>
    <%= stylesheet_tag 'blob' %>
  </head>
```

```
  <body>
    <%= yield %>  # the content will show up here
  </body>
</html>

---
class MyController < ApplicationController
  layout :blog, :except => [:rss, :atom]
  ...
end
---
class MyOtherController < ApplicationController
  layout :compute_layout

  # this method computes the name of the layout to use
  def compute_layout
    return 'admin' if session[:role] == 'admin'
    'standard'
  end
  ...
end
```

Layouts have access to the instance variables of the controller.

Partials

Partials a building blocks for creating views. They allow you to reuse commonly used display blocks. Partials are stored in files:

```
render :partial => 'product'
```

This command loads the partial stored in *_product.rhtml* and passes the instance variable *@product* to it. The partial can access it using *@product*.

```
render :partial => 'product', :locals => { :product => @bought }
```

This command loads the same partial but assigns a different instance variable to it:

```
render :partial => 'product', :collection => @product_list
```

This renders the partial for each element in *@product_list* and assigns *@product* to each element.

6 Ajax

Be sure to include the JavaScipt libraries in the layout:

```
<%= javascript_include_tag :defaults %>
```

6.1 Linking to a Remote Action

```
<%= link_to_remote "link", :update => 'some_div',
                           :url => { :action =>'show', :id => post.id } %>

<%= link_to_remote "link", :url => { :action =>'create',
                           :update => { :success => 'good_div',
                                        :failure => 'error_div' },
                           :loading => 'Element.show('spinner'),
                           :complete => 'Element.hide('spinner') } %>
```

6.2 Callbacks

:loading Called when the remote document is being loaded with data by the browser.

:loaded Called when the browser has finished loading the remote document.

:interactive Called when the user can interact with the remote document, even though it has not finished loading.

:success Called when the XMLHttpRequest is completed, and the HTTP status code is in the 2XX range.

:failure Called when the XMLHttpRequest is completed, and the HTTP status code is not in the 2XX range.

:complete Called when the XMLHttpRequest is complete (fires after success/failure if they are present).

6.3 Ajax Forms

You can create a form that will submit via XMLHttpRequest instead of a POST request. The parameters are passed exactly the same way (so the controller can user the *params* method to access the parameters). Fallback for non-JavaScript-enabled browsers can be specified by using the *:action* methods in the *:html* option:

```
<%= form_remote_tag :html => { :action => url_for(:controller => 'controller',
                                                  :action => 'action'),
                               :method => :post }
</form>
```

generates:

```
<form action="/post/save" method="post" onsubmit="new Ajax.Request(",
{asynchronous:true, evalScripts:true, parameters:Form.serialize(this)}); return false;">
</form>
```

6.3.1 Autocompleting Text Field

In the view template:

```
<%= text_field_with_auth_complete :model, :attribute %>
```

In the controller:

```
auto_complete_for :model, :attribute %>
```

6.3.2 Observe Field

```
<%= start_form_tag ( {:action => "save"},
                     :id =>"entry-form") %>
Title <br>
<%= text_field :entry, :title %>
<br>
Body<br>
<%= text_area :entry, :body %>
<br>
<%= submit_tag "Save" %>
<%= end_form_tag %>
<%= observe_form 'entry-form',
            :frequency => 1,
            :update => "live-preview",
            :complete => "Element.show('live-preview')",
            :url => {:action => 'preview' }
```

```
            %>

<div id='live-preview' ></div>
```

generates:

```
<form action="/diary/save" id="entry-form" method="post">
Title <br>
<input id="entry_title" name="entry[title]" size="30" type="text" />
<br>
Body<br>
<textarea cols="40" id="entry_body" name="entry[body]" rows="20"></textarea>
<br>
<input name="commit" type="submit" value="Save" />
</form>
<script type="text/javascript">
//<![CDATA[
new Form.Observer('entry-form', 1, function(element, value) {new Ajax.Updater('live-preview',
'/diary/preview', {asynchronous:true, evalScripts:true,
onComplete:function(request){Element.show('live-preview')}, parameters:value})})
//]]>
</script>

<div id='live-preview' style='display: none; border: 1px solid'></div>
```

6.3.3 Observe Form

Same semantics as *observe_field*.

7 Configuring Your Application

The main configuration file resides in *config/environment.rb.* This list contains only a few of the possible options.

7.1 Session Configuration

```
# store session data in database (requires sessions table: rake db:sessions:create; rake db:migrate)
config.action_controller.session_store = :active_record_store
```

7.2 Email Configuration (SMTP)

Email configuration is done the *config/environments/** files.

Example SMTP configuration:

```
config.action_mailer.server_settings = {
  :address => "smtp.sbcglobal.yahoo.com",
  :port => 25,
  :domain => "smtp.sbcglobal.yahoo.com",
  :authentication => :login,
  :user_name => "Your_User_Name@sbcglobal.net",
  :password => "Your_Password"
}
```

That's All for Now!

Rails is a large framework. We've covered many of the important features in this little guide. Keep in mind that Rails is ever-evolving, but as you use it for your applications, you will become more comfortable with it.

Keep hacking...

–Doug Putnam

All of the material in this lexicon is adapted from *Ruby on Rails — Up and Running*, By Bruce Tate & Curt Hibbs