

Chapter 1: An Overview Of Ruby on Rails

1) What is Ruby on Rails?	1-2
2) Downloading Rails	1-3
3) A Rails Application	1-4
4) Structure of a Rails Application	1-7
5) Building the Controller.....	1-8
6) Views	1-13
7) A Quick Review.....	1-15
8) Rails Conventions.....	1-16
9) Embedded Ruby.....	1-18
10) Extended Time Functions	1-21
11) The render Method	1-23
12) The link_to Method.....	1-24
13) One Last Thought	1-25

Evaluation
Copy

What is Ruby on Rails?

- What is Rails? To answer the question, we turn to the Wikipedia definition of Ruby on Rails.
 - ▶ **Ruby on Rails** is a web application framework released in 2004 that aims to increase the speed and ease of web development.
 - ▶ Often shortened to **Rails** or **RoR**, it is an open source project written in the Ruby language.
- A **web application framework** is a set of software tools and libraries that make it easier to create web applications.
 - ▶ They typically provide functionality such as database access, templating, and session management.
 - ▶ A template is a way of providing output to a browser.
 - ▶ A session refers to the time a user utilizes a browser to "shop."
- Rails uses the model-view-controller design pattern.
 - ▶ The model is the **domain**-specific representation of the data on which the application operates. The model is responsible for validating data and specifying relationships for data, etc. Domain-specific logic is also handled in the model.
 - ▶ The view is responsible for rendering the data, usually in the form of a user interface. In web applications, the view is expressed as an HTML page.
 - ▶ The controller processes and responds to requests from the browser. The controller is the focal point of the application.

Downloading Rails

- When you install Ruby, you also get some related tools. One such tool is the `gem` application. This is used to interface with the RubyGems package system.
- RubyGems is a system for managing Ruby software libraries. Ruby code packaged in this manner is called a `gem`.
- When you find Ruby software that you wish to use in a project, `gem` offers a means of downloading, installing, and managing the software.

- The RubyGems system is similar to using CPAN in the Perl world. Typically, the `gem` command is available as part of the Ruby download.

- You can use `gem` to install Rails. Simply type:

```
$ gem install rails --include-dependencies
```

- If you actually executed the above step, you would see a lot of output on your display. The numbers represent the version of the installed software.
- The output refers to exactly what was installed in support of Rails. Once you have installed Rails, you can begin your first web application (project).

A Rails Application

- In order to see what a Rails application looks like, we will build a simple one. As you proceed, you will notice that Rails favors the use of convention over configuration.
- Rather than having a preponderance of XML files, Rails favors the use of naming conventions. You will see this as we move forward.
- So that everybody in the course will have the same directory structure, create a directory named `railsapps`, and then `cd` there.

```
$ mkdir railsapps  
$ cd railsapps
```

- Note that throughout the course, we will show the Unix Shell prompt (`$`), but we also could have used the Windows DOS command prompt (`C:\>`).

- Now enter the following.

```
$ rails first
```

- We are calling this application `first`. You should see a lot of output. Each line of output is the name of a file or directory in the tree structure built to support the application that we have named `first`.

A Rails Application

- If you wish to get an idea of what is contained in the directories that we just created, examine the README file in the railsapps/first directory.

- In Rails 2.x, the default database is `sqlite-3`. If you wish to configure your database for `MySQL` instead, then you will need to build your skeleton application with:

```
$ rails -d mysql first
```

- Of the directories created, one of them is named `script`. One of the files in this directory is a Ruby script named `server`, which launches the `WEBrick` web server.

- While in the `first` directory, you can start this server.

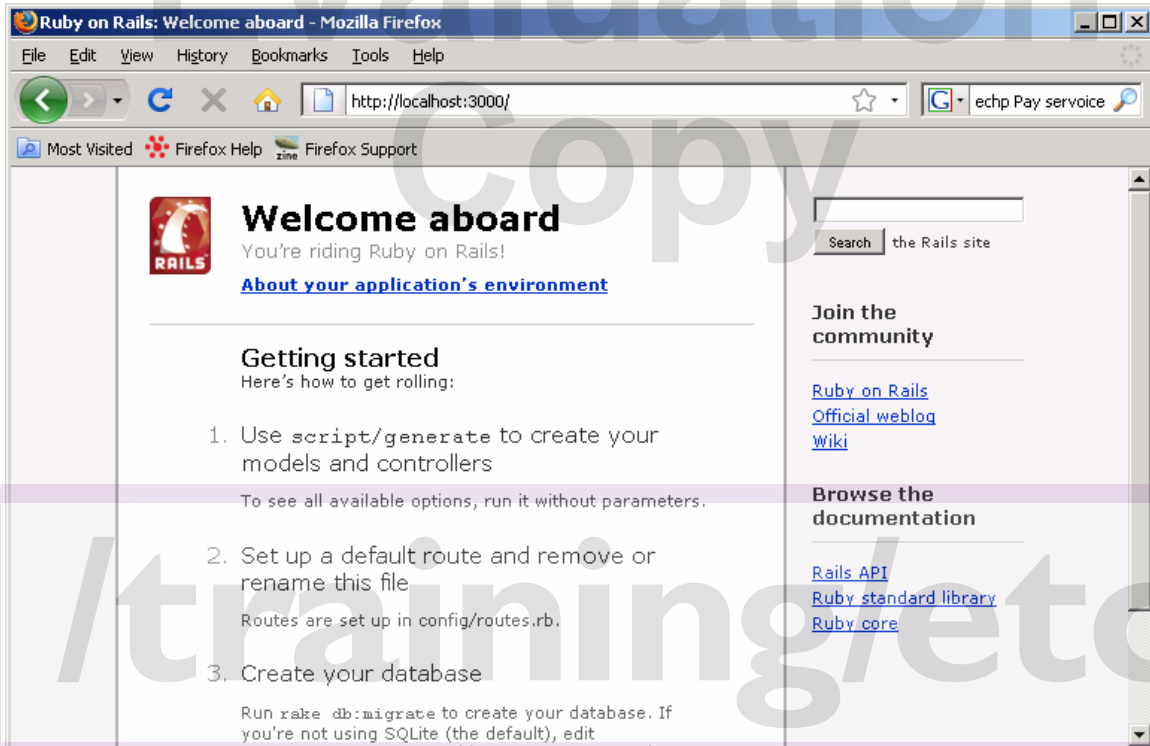
```
$ ruby script/server
=> Booting WEBrick...
=> Rails 2.1.0 application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2008-09-29 13:03:10] INFO WEBrick 1.3.1
[2008-09-29 13:03:10] INFO ruby 1.8.6 (2007-03-13) [i386-
mswin32]
[2008-09-29 13:03:10] INFO WEBrick::HTTPServer#start: pid=2728
port=3000
$
```

- We will use this server as we proceed through the course. However, we could have used other servers such as `Apache`.
- If you want to ensure the server is up and running, access it on port `3000` by typing the following URL in your Browser's address box.

```
http://localhost:3000
```

A Rails Application

- By doing this, you should see the following picture.



- Now that we know the server is running, we will demonstrate a Web Application at work.

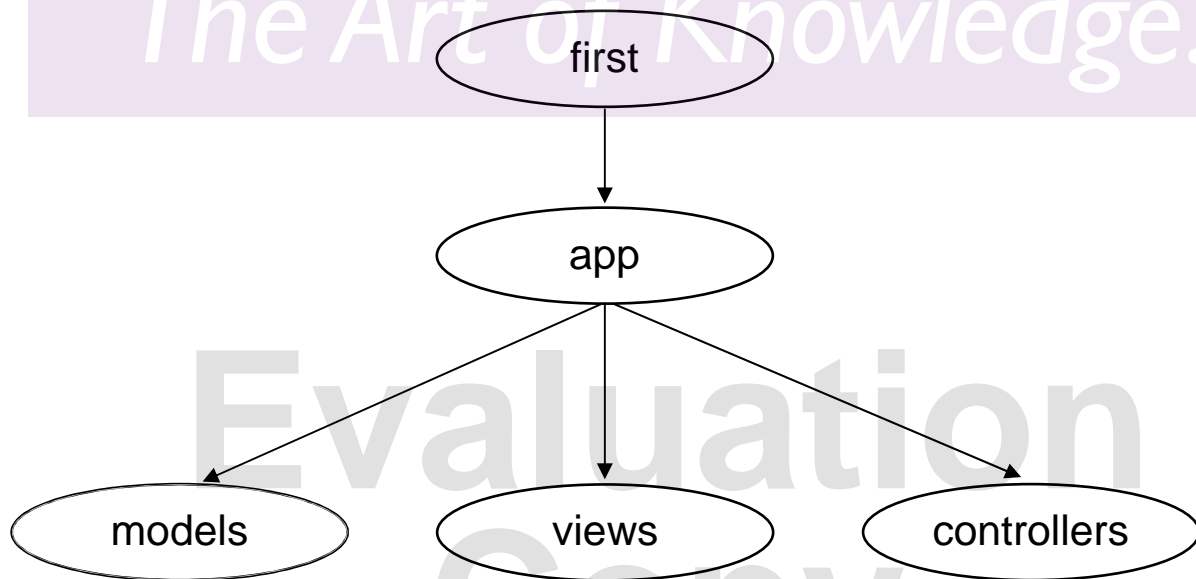
- First, we need to describe a portion of the MVC architecture. In particular, we need to give some information about the controller.
- In order to launch a web application, the user does so through a URL. For a Rails application, a URL maps to the name of a controller that lives in the application's `app/controllers` directory.

Structure of a Rails Application

- When you used the `rails` command to build the skeleton of the application, there were many files and directories created for you.
- We will explore most of these directories and files as we learn more about building web applications with Rails.
- The directory structure below `first` looks like this.

```
app      config      db
doc      lib         log
public  script      test
vendor  tmp
```

- Each directory has its purpose as we will explore during the course. For now, we concentrate on the `app` directory, which has three principal subdirectories.



Building the Controller

- In Rails, the controller is a file generated by a Ruby script. You can generate this file with the following command. Make sure you are in the `first` directory.

```
$ ruby script/generate controller Test
exists  app/controllers/
exists  app/helpers/
create  app/views/test
exists  test/functional/
create  app/controllers/test_controller.rb
create  test/functional/test_controller_test.rb
create  app/helpers/test_helper.rb
$
```

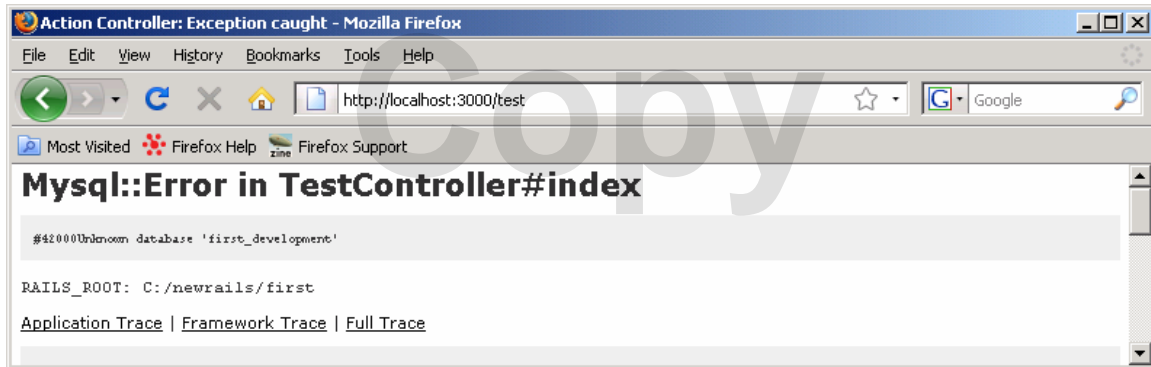
- In this application we have called our controller `Test`. We could have chosen any name. Notice that several files have been created by the `generate` script.
- We will discuss these later, but for now we need to discuss the controller itself. As you will see, the controller is a class whose name is `TestController`. This class lives in the file `test_controller.rb`.

- To launch the application, append the name of the controller (`test` in this case) to the URL for the application.

```
http://localhost:3000/test
```

Building the Controller

- When you navigate to the URL above, you can see that an error message is generated.

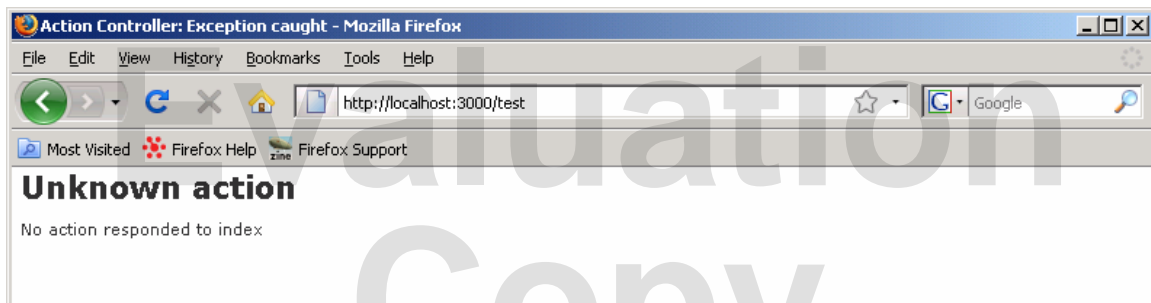


- Rails is thinking ahead. It is assuming that you will eventually want to deal with a database. For now, we will create a database named `first_development`, and we will fill in the explanations for this step later.

- How you create the database is up to you. One simple way is to use the `mysqladmin` tool.

```
$ mysqladmin -u root create first_development
```

- After you have added the database, try the same URL once again. Unfortunately, this time you will get another error message.



Building the Controller

- As you can see, the error message below tells you that you have a missing `index` method.

Unknown action

No action responded to `index`

- You may not be aware of this, but this error message is at the heart of the Rails paradigm. When you enter a URL in the Rails paradigm, a request is being made.
- Rails carries out certain steps in order to build a response to that request. The steps are listed below.
 - ▶ Find the `test` controller in the `app/controllers` directory of your application.
 - ▶ Find the `index` method in this `Test_Controller` class.
- Rails is trying to find a method called `index` in the controller class. In Rails parlance, an action is equivalent to a method in the controller.
- Generally speaking, the set of actions in the controller defines the behavior of this application. This is not entirely true since an application is likely to have more than one controller, each of which controls a certain functional segment of an application.

Building the Controller

- At this point, the controller class looks like this.

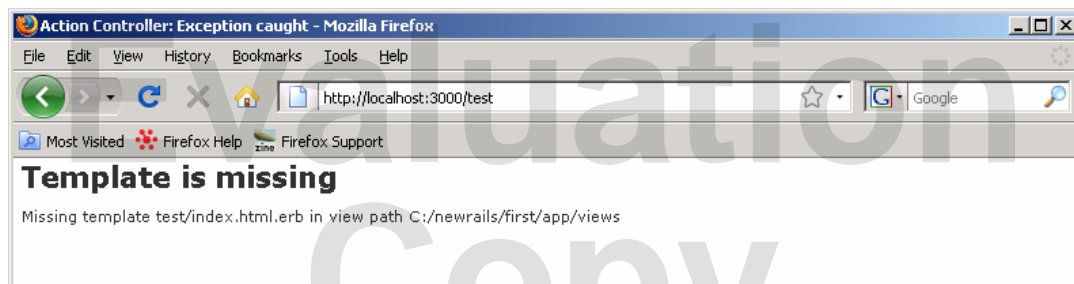
```
class TestController < ApplicationController
end
```

- The class `ApplicationController` provides the necessary inherited methods for a controller.
- Note also that there is another file in the directory named `app/controllers`. This file named `application.rb` is more global in nature, and its role will be discussed later.

- For now, to eliminate the Browser's error message, we need to provide the `index` method. Therefore, we modify the controller as shown below.

```
class TestController < ApplicationController
  def index
    puts "inside the Controller"
  end
end
```

- If you rerun the application now, you will see yet another error message. In addition the "puts" will be displayed within the server's console.



Building the Controller

- This time the error message is:

Template is missing

Missing template test/index.html.erb in view path C:/newrails/first/app/views

- We are being warned about the lack of a template. Now, we need to know the rest of the Rails paradigm.
- Rails carries out certain steps in order to build a response to a request. The steps follow.

- ▶ Find the `test` controller in the `app/controllers` directory of your application.
- ▶ Find the `index` method in this `Test_Controller` class.
- ▶ Render the response to the original request by "executing" the template file found in `app/views/test` and named `index.html.erb`.

- Note that the name of the template file is taken from the name of the action (i.e., the `index` action is executed, and Rails then looks for the `index.html.erb` file to render a result).
- To finally see a result, we must create the template file `index.html.erb`.
- A template is another name for a view. The view file actually causes something to be displayed in the browser.

Views

- The problem above can be solved by creating a `view`. A view is a file that is associated with an action.
- It is the view that gives the controller the means to display the results of an application.
- A view is a template for displaying things. It is like an HTML file. Later, we will see that instance data from each controller method is accessible from the view corresponding to that method.
- A view file is a webpage named with respect to an action in a controller. It is named with a suffix as shown here.
 - ▶ `.rhtml` (pre Rails 2.0)
 - ▶ `.html.erb` (Rails 2.0 and up)
- We will name the view for this `index` method `index.html.erb` and place this file in the `app/views/test` directory. As an example, the file `index.html.erb` might appear as shown below.

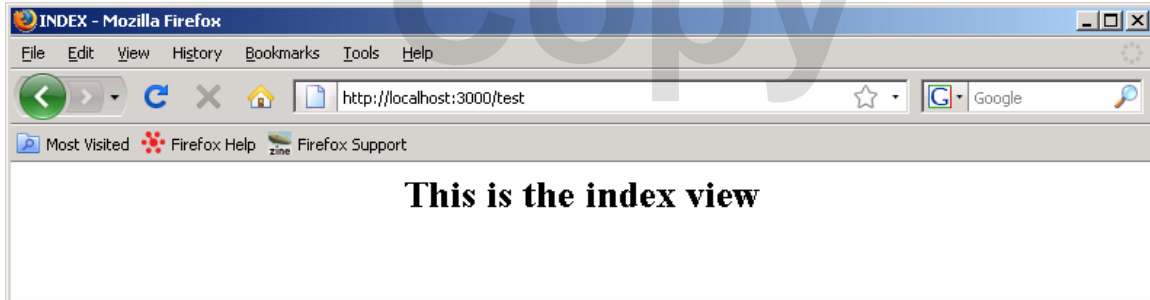
```
<html>
<head>
<title>
INDEX
</title>
</head>
<body>
<h1><center>This is the index view</center></h1>
</body>
</html>
```

Views

- Now if you navigate to:

`http://localhost:3000/test`

you will see the following rendering.



- The template, `index.html.erb`, could have been written as:

```
<h1><center>This is the index view</center></h1>
```

- Template files are ones that contain:

- ▶ HTML code; and
- ▶ embedded Ruby.

- Even though our course notes may reflect some incomplete HTML syntax, it is a strong recommendation that your HTML syntax conform to the current XHTML standard.
- At this point, we should review some of the conventions that Rails enforces.

A Quick Review

- A Rails application is created with the rails command.

```
$ rails -d mysql myapp
```

- Many scripts exist in the `myapp/script` directory. These scripts will help us in building some skeleton code for any Rails application. A few examples of these scripts are shown below.

- ▶ Start the server.

```
$ ruby script/server
```

- ▶ Create a controller.

```
$ ruby script/generate controller School
```

- Rails applications are launched by requesting a URL from a browser.

- The URL contains the name of a controller followed by the name of an action. If no action is given, then the `index` action is assumed. Therefore, the following two URLs are equivalent.

```
http://localhost:3000/test  
http://localhost:3000/test/index
```

- Once an action is found, it is executed, and following its execution, a template is rendered. The template for controller `X` lives in the directory `app/views/X` and bears the name of the action in the controller.
- Therefore, for the action `Y`, the template will have the name `Y.html.erb` or `Y.rhtml`.

Rails Conventions

- Rails applications typically have more than one controller, although the application that we are building, named `first`, has only one controller.
- Each controller typically has many actions, each of them representing one behavior for that controller.
- For example, to add a second action to the `test` controller, we would do the following.
 - ▶ Modify the controller itself.

```
class TestController < ApplicationController
  def index
    puts "inside the Controller"
  end

  def tryme
    puts "inside the tryme method"
  end
end
```

- ▶ Create a `tryme.html.erb` file in the `app/views/test` directory.

```
<html><head><title>
TRYME
</title></head><body>
<h1><center>This is the tryme view
</center></h1>
</body></html>
```

Rails Conventions

- Now, notice what happens when we enter the following.

```
http://localhost:3000/test/tryme
```



- At this point, the following files would be relevant.

```
app
  controllers
    hello_controller.rb
  models
  views
test
  index.html.erb
  tryme.html.erb
```

- Keep in mind that in real applications, the methods in the controller would have Ruby code within them. However, we are simply learning the Rails paradigm at this stage of the course.

Evaluation
Copy

Embedded Ruby

- It is important to know why the file name for a template must end in `.erb` (or `.rhtml`).
- `.erb` (and `.rhtml`) files are similar to `.asp` (Active Server Pages) or `.jsp` (Java Server Pages) files. An `.asp` file will have HTML with embedded Visual Basic code. A `.jsp` file will have HTML with embedded Java code.
- Therefore, an `.erb` file or an `.rhtml` file will have HTML with embedded Ruby code.

- The embedded ruby code is surrounded by the following special tags.

```
<%= RUBY CODE GOES HERE %>
```

- Of course, there can be many instances of these special tags in any `.rhtml` file.

- There is another set of special tags that look very much like those above. (There is no `=` sign however.)

```
<% RUBY CODE GOES HERE %>
```

- The first set of tags will cause the output of the Ruby code to be inserted into the HTML stream. The second set of tags will cause the Ruby code to be evaluated but not placed in the resultant HTML stream.

Embedded Ruby

- Below is an example using each of these special tags. We will create another action and template to demonstrate these tags. We will name the action `evaluate`. The controller now looks like this.

```
class TestController < ApplicationController
  def index
    puts "inside the controller"
  end
  def tryme
    puts "inside the tryme method"
  end
  def evaluate
    puts "inside the evaluate method"
  end
end
```

- The template file will be called `evaluate.html.erb`.

```
<html><head><title>EVALUATE</title>
</head><body><h1>
<center>This is the evaluate view</center>
<% x = 2 + 2 %>
<center>2 + 2 = <%= x %></center>
<center>Today is <%= Time.new %></center>
</h1></body></html>
```

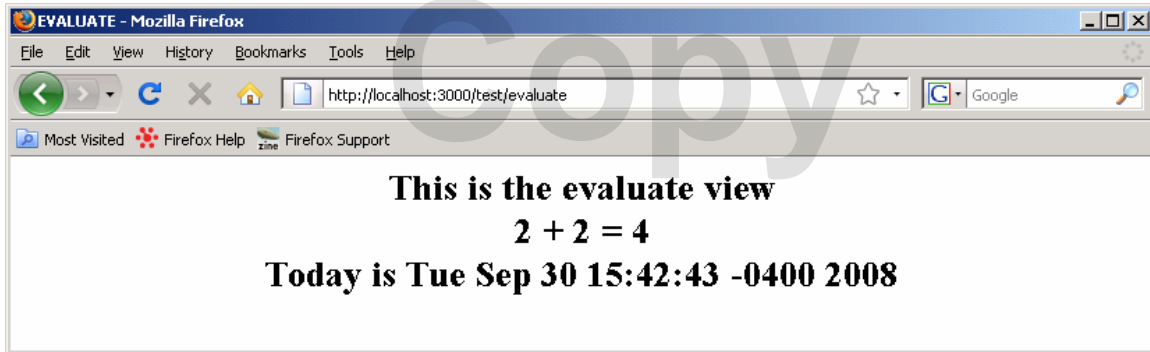
- A word of caution is necessary here. When you use the Ruby `puts` function within the special tags, the results are sent to the WEBrick console.

```
<%= puts "string" %>
```

Embedded Ruby

- Below is the rendering of the following URL.

`http://localhost:3000/test/evaluate`



- In the previous example, the "calculation" $2 + 2$, was made in the view file. Since a template is aware of instance data within its respective controller action, we could have put the logic in the controller and let the view simply display it.

- To demonstrate this, we will write another method and another `.erb` file. We will call it `localeval`.

```
def localeval
  @data = 2 + 2
  @time = Time.now
end
```

- The template `localeval.html.erb` might be:

```
<h1><center>This is the localeval view</center>
<br><center> 2 + 2 = <%= @data %></center>
<br><center>Today is <%= @time %></center></h1>
```

Extended Time Functions

- In the last section, we saw a few instances of the time being displayed. Rails contains extensions to the basic time functions. Below are a few that you might find useful. To demonstrate, we will create a new action, `mytime`.

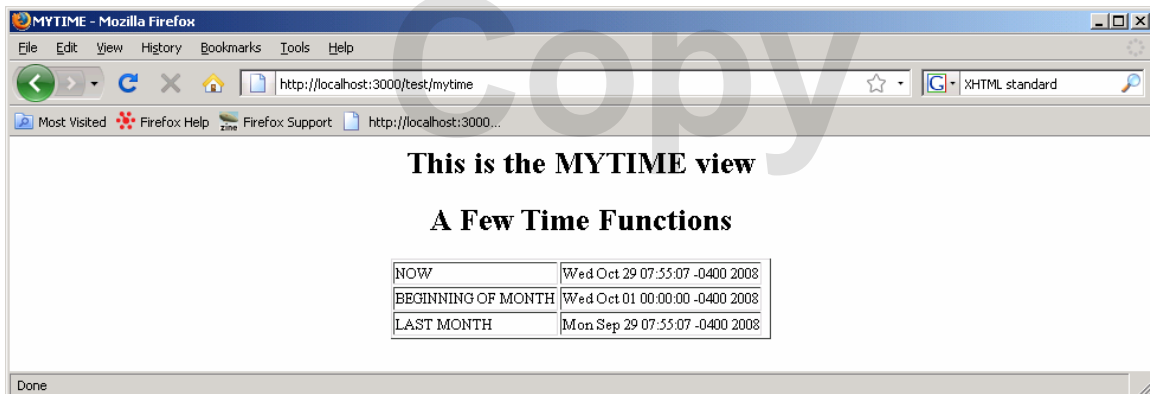
```
def mytime
  @now = Time.now
end
```

- Below is the template (`mytime.html.erb`) and the display.

```
<html>
<head>
<title>MYTIME</title>
<head>
<body>
<h1><center>This is the MYTIME view</center>
</h1>
<h1><center>A Few Time Functions</h1></center>
<center><table border="2">
<tr><td>NOW</td><td><%= @now %></td></tr>
<tr><td>BEGINNING OF MONTH</td>
      <td><%= @now.at_beginning_of_month %></td>
</tr>
<tr><td>LAST MONTH</td>
      <td><%= @now.last_month %><td>
</tr></table></center>
</body>
</html>
```

Extended Time Functions

- The output for the rendering of the `mytime` view is shown below.



/training/etc

The Art of Knowledge.

Evaluation
Copy

The `render` Method

- We have yet to see the power of Ruby. The controller methods seen so far have been simple, so that we could illustrate the basic Rails paradigm.
 - ▶ An application is launched through a URL containing a controller and an action within it.
 - ▶ The action in the controller is executed.
 - ▶ The template for that action is automatically rendered.

- In real applications, Ruby code is used within these actions. Depending upon the results of that code, either the template for the action is rendered or you can render another template.

- The rendering of a template is automatic when all the code in the method has been executed. But you can explicitly use the `render` method if you wish to render another template.

```
render(:action => "tryme")
```

- The `render` method tells Rails which view to display. In the above case, `tryme.html.erb` would be rendered. In the `render` method, you do not need to encode the suffix (`.html.erb` or `.rhtml`).

- You could also render text.

```
render(:text => "you forgot to enter data")
```

The `link_to` Method

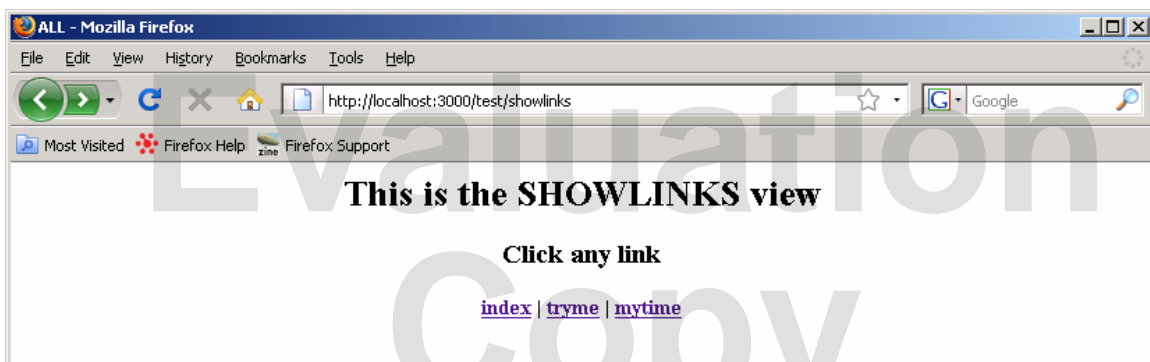
- The `link_to` method allows you to navigate easily to other pages in your application.

```
link_to 'The Link', :action => "some method"
```

- The first argument to this method is the hypertext itself. The second argument is the method (action) that will be called when you select the link.
- For example, suppose we create the `showlinks` action and provide the `showlinks.html.erb` template as shown here.

```
<html><head><title>ALL</title></head><body>
<h1><center>This is the SHOWLINKS view
</center></h1>
<h2><center>Click any link</center></h2>
<h3><center>
<%= link_to 'index', :action=>'index'%> |
<%= link_to 'tryme', :action=> 'tryme'%> |
<%= link_to 'mytime', :action=> 'mytime' %> |
</body></html>
```

- When you "hit" the `showlinks` action, you will see the following. Notice that there are three hot links!



One Last Thought

- Throughout this course, as you learn more about producing database backed Web Applications with Rails, you will encounter three major libraries that can be viewed as the backbone of any Rails application.
- One of these libraries is the `ActiveRecord` library. This library is responsible for both:
 - ▶ writing Ruby objects to a database table; and
 - ▶ retrieving rows from a database table and storing them in Ruby objects.
- Another library is the `ActionPack` library. This contains two sub-libraries.
 - ▶ `ActionView`
 - ▶ `ActionController`
- `ActionController` supports the code you write in your controllers.
- `ActionView` processes the code in the view templates and generates HTML code from them.

Evaluation
Copy

Exercises

1. You are going to mimic most of the things we have done in this chapter.
 - (a) Create a Rails application named `solution1`.
 - (b) Create a controller named `Store`.
 - (c) Create a few actions: `index`, `buy`, and `sell`.
 - (d) Create the templates for each of these actions.
 - (e) Start the Webrick server.
 - (f) Test each of the following URLs. Do not forget to create the `solution1_development` database.

```
http://localhost:3000
http://localhost:3000/Store
http://localhost:3000/Store/index
http://localhost:3000/Store/buy
http://localhost:3000/Store/sell
```

- (g) In your `index` action, generate a random number between 0 and 99.

```
rand(100)
```

- (h) Now use the `render` method to render either the `buy` or `sell` template, depending upon whether the random number is greater than 50. In either case, print the random number from your template.
 - (i) Now, place a link in both the `buy` and `sell` templates that will allow a user to easily navigate back to the `index` method.