Prepared exclusively for Merb course participants

Introduction to Merb

Revised Edition – 17th Jan. 2009
First Edition – Nov. 2008

PLEASE SUPPORT RUBYLEARNING.COM

DONATE pledgie.com
$106.00 Raised!

Table of Contents

Acknowledgements

There are a good number of people who deserve thanks for their help and support they provided, either while or before this eBook was written, and there are still others whose help will come after the eBook is released.

I'd like to thank my ‘gang’ of assistant teachers, mentors and patrons at http://rubylearning.org/ for their help in making these study notes far better than I could have done alone.

I have made extensive references to:

- information, related to Merb, available in the public domain (wikis and the blogs, articles of various **Ruby Merb Gurus**)
- the following eBooks:
  Merb in Action
  http://manning.com/ivey/
  The Merb Way
  http://safari.oreilly.com/9780321601636

My acknowledgment and thanks to all of them.

Thanks to:

Matt Aimonetti (Merb Core Developer)
http://www.workingwithrails.com/person/6065-matt-aimonetti
and **Yehuda Katz** (Merb Maintainer)
http://www.workingwithrails.com/person/1805-yehuda-katz
for suggesting the topics to our “**Introduction to Merb**” course at RubyLearning.

Also, special thanks to **George Thompson** and **Michèle Garoche** who helped out extensively on some of the topics in this eBook.

Introduction

Assumptions

1. I am assuming that you know Core Ruby
2. You understand how Ruby Gems work
3. Have a basic understanding of the MVC paradigm – http://en.wikipedia.org/wiki/Model-view-controller

Concept and Approach

I have tried to organize this eBook in which each chapter builds upon the skills acquired in the previous chapter, so that you will never be called upon to do something that you have not already learned. This eBook not only teaches you how to do something, but also provides you with the chance to put those morsels of knowledge into practice with exercises. I have therefore included several exercises, or assignments, in this eBook so that you will have opportunities to apply your knowledge.

How to Use This Merb eBook

I recommend that you go through the entire Merb eBook chapter by chapter, reading the text, running the sample programs and doing the assignments along the way. There are no large applications in this book – just small, self-contained sample programs. This will give you a much broader understanding of how things are done (and of how you can get things done), and it will reduce the chance of anxiety, confusion, and worse yet, mistakes.

It is best to read this eBook and complete its assignments when you are relaxed and have time to spare. Nothing makes things go wrong more than working in a rush. And keep in mind that Merb and the assignments in this eBook are fun, not just work exercises. So go ahead and enjoy it.

About the Conventions Used in This Merb eBook

Explanatory notes (generally provides some hints or gives a more in-depth explanation of some point mentioned in the text) are shown shaded like this:

This is an explanatory note. You can skip it if you like - but if you do so, you may miss something of interest!

Any source code in this Merb eBook, is written like this:

```
def hello
  puts “hello”
end
```

When there is a sample program to accompany the code, the program name is shown like this: **`hello.rb`**

Though we would be discussing Merb on the Windows platform, these notes/course is appropriate for Linux/Mac users as well.

If you notice any errors or typos, or have any comments or suggestions or good exercises I could include, please email me at: **mail@satishtalim.com**.

Why Merb?

Choosing a framework is the first and most important decision anyone building a web application will make. A poorly matched framework will bring devastating roadblocks and a rewrite may cost you the farm. You owe it to yourself to pick the right framework, and if you're serious about your application, then Merb is the framework that can grow with your needs – Author *Foy Savas* of The Merb Way

Merb is potentially much faster than Rails, and with the performance improvements to Ruby (C-Ruby, JRuby, etc.), the combination of Merb and Ruby may well provide adequate performance for most of my new projects in the future. Merb has more to offer than just runtime performance. For example, its small core of functionality with plugins and "slices" (complete mini-apps that install in an existing Merb project) does not, in principle, have cross dependencies. Because Merb was designed to be modular, you can include just the plugins and "slices" your application needs, keeping your systems simpler.

See http://merbcamp.com/video/neighman1.mp4 for more information on slices.

Perhaps more importantly, Merb is thread safe and has built-in support for running background tasks as threads. That means deployed Merb applications are likely to require much less memory because you can use work threads instead of extra processes to serve up page and web service requests. Without this feature, you risk duplicating application and framework libraries (if shared libraries are not used) as well as application data when you run many identical processes on a server. (**Reference**: http://www.devx.com/webdev/Article/39735)

Also note: Please do not try out the example given in the above url (it uses Merb prior to version 1.0), and a lot has changed in Merb (currently at 1.0.7.1).

Comparing a Rails app with ActiveRecord and a rewrite with Merb/DataMapper, most people will find Merb/DataMapper to be 5x+

faster. One needs a thread-safe ORM like DataMapper to take advantage of this.
(**Reference**: http://railspikes.com/2008/6/6/merb-and-rails-interview-with-ezra)

Finally, in another 6-8 months Merb 2.0 will be Rails 3.0. With the Rails and Merb merger, this makes the "Introduction to Merb" training at RubyLearning, basically the first training focusing on the future features of Rails 3, and a great way to get ahead of the game.

What's Merb?

Merb (currently at version 1.0.7.1) is the brainchild of Ezra Zygmuntowiczis and is a Model View Controller (MVC) framework written in Ruby and used for building fast, scalable database-driven web applications.

Merb adopts an approach that focuses on essential core functionality, leaving most functionality to plugins. Core framework functionality was modularized into "*merb-core*", and all extra functionality (e.g. various templating language plugins) inside a "*merb-more*" and "*merb-plugins*" components.

Merb allows for developer preference in choice of ORM (ActiveRecord, DataMapper (http://datamapper.org/doku.php), and Sequel).

Merb supports any web server including Mongrel, Evented Mongrel, Thin, FCGI, CGI, and even Webrick.

Some Resources:
Merbivore is the official Merb site –
http://www.merbivore.com/

Latest Merb Core API Documentation –
http://www.merbivore.com/documentation/current/doc/rdoc/merb-core/index.html

Merb's official Blog -
http://planet.merbivore.com/

Merb discussion forum -
http://groups.google.com/group/merb

Ezra Zygmuntowiczis talks to RubyLearning –
http://rubylearning.com/blog/2008/12/02/little-known-ways-to-ruby-mastery-by-ezra-zygmuntowicz/

Matt Aimonetti: Why on earth would you ignore Merb?
http://rubylearning.com/blog/2008/12/18/matt-aimonetti-why-on-earth-would-you-ignore-merb/

Yehuda Katz talks to RubyLearning –
http://rubylearning.com/blog/2008/12/23/little-known-ways-to-ruby-mastery-by-yehuda-katz/

Reference: http://en.wikipedia.org/wiki/Merb

Installation

We shall be discussing the installation procedures for –

- Merb 1.0.7.1
- SQLite 3.6.7
- Mongrel 1.1.5

I am using the following configuration –

Windows XP with SP2 box
Ruby 1.8.6 (2007-09-24 patchlevel 111) [i386-mswin32]
Ruby Gem 1.3.1
Merb 1.0.7.1
SQLite 3.6.7
do_sqlite3-0.9.9
Mongrel 1.1.5

Merb Installation

a. **Install Ruby 1.8.6**: Getting up and running with Merb requires that you have a basic installation of Ruby 1.8.6. Once your Ruby environment is set up, it's time to install Merb.

b. **Installing Merb**: We simply install Merb from gems. Make sure that you are running RubyGems 1.3.0 or higher. To check, run:

```
gem –v
```

Note: If not, you need to upgrade your gem (The page - http://www.ruby-forum.com/topic/166853#732427 is very useful for doing that).

For Windows:
Windows user needs to install **`do_sqlite3`** before installing merb.
Download the **`do_sqlite3`** gem from –
`http://rubyforge.org/frs/download.php/49591/do_sqlite3-0.9.10.1-x86-mswin32.gem`

Copy the downloaded file **do_sqlite3-0.9.10.1-x86-mswin32.gem** to a **temp** folder and from this **temp** folder, and type:

```
c:\Temp> gem install do_sqlite3-0.9.10.1-x86-mswin32.gem
Successfully installed do_sqlite3-0.9.10.1-x86-mswin32
1 gem installed
c:\Temp>
```

Next, the command to install Merb is:

```
c:\> gem install merb
Successfully installed merb-1.0.7.1
1 gem installed
c:\>
```

Also, Windows users may be asked to install **dm-core** gem. The command to install the gem is:

```
c:\> gem install dm-core
Successfully installed dm-core-0.9.9
1 gem installed
Installing ri documentation for dm-core-0.9.9...
Installing RDoc documentation for dm-core-0.9.9...
c:\>
```

It is suggested that you also install webrat, as it is required for interactive merb. To install type:

```
c:\> gem install webrat
c:\>
```

If you're on a **Unix**, you likely need to prefix with sudo, as in **sudo gem install merb**. You will see a large number of gems install. This is normal. As we said earlier, Merb is made up of a number of modules, and **gem install merb** will automatically download and install all of them for you.

If you get an error while installing some of the documentation, then do:
gem install merb --no-ri --no-rdoc

It seems the comments used to generate the documentation cannot be processed properly by ri. Just install it without docs and all will be fine.

Note 1: To check whether a gem has been successfully installed, you can type:

```
c:\> gem specification merb
```

you will see all the details of the “merb” gem.

Note 2: merb is a meta-gem for the "Full Stack". It includes everything you need to get working. You can find out the full list of dependencies for the merb gem, by typing "**`gem dependency merb`**” at the command window. The full stack meta-gem includes merb-core, merb-more, do_sqlite3, datamapper, and all the bits in between to make them fit together, and then some more fun stuff.

Note 3: If you experience any problem while installing, Google is your friend. You will typically be able to find someone else who has experienced exactly the same problem as you. You can also stop by the #merb channel in IRC on irc.freenode.net. People in the IRC room are friendly, patient, and will likely know something about the issue you’re having.

Instructions for Ubuntu 8.10 (Intrepid Ibex) O/S:
(Thanks to **Mark DeWandel**)

Ubuntu 8.10 comes with RubyGems 1.2.0 installed. However, in order to install Merb you will need RubyGems 1.3.1 as mentioned here – http://www.ruby-forum.com/topic/166853#732427

Roll the data_objects and do_sqlite3 gems back to version 0.9.7 from 0.9.8:

```
sudo gem uninstall data_objects --version 0.9.8
sudo gem install data_objects --version 0.9.7 --include-
dependencies

sudo gem uninstall do_sqlite3 --version 0.9.8
sudo gem install do_sqlite3 --version 0.9.7 --include-
dependencies
```

Latest Installation Instructions -
http://book.merbist.com./getting-started/install-instructions

Updating Merb and the ruby_bookshop project

Merb is being updated nearly every week and you would need to update your merb installation. So if you are updating Merb from 1.0.6 to 1.0.7.1 then do the following:

1. Type: `c:\>` **`gem update merb`**

2. Change version numbers in *all your app's* **`dependencies.rb`** to:

```
merb_gems_version = "1.0.7.1"
dm_gems_version   = "0.9.9"
```

SQLite Installation

We need to decide what database we want to use. We'll use **SQLite version 3.6.x**, which comes pre-installed on OSX (**Note**: On a Mac OS X 10.4.11 PPC, the **Sqlite 3** version installed is 3.1.3, so one has to install a newer version) and many flavours of Unix).

You can check whether it's installed by typing **`sqlite3`** on the command line, which will bring you to a prompt marked **`sqlite>`** if it is installed. If not, install it via your package manager, or if you're on Windows, by downloading the binary from the SQLite website - http://www.sqlite.org/

Full installation details are available here –
http://wiki.rubyonrails.org/rails/pages/HowtoUseSQLite

SQLite FAQ –
http://www.sqlite.org/faq.html

For Windows XP, I have downloaded the following "*Precompiled Binaries For Windows*" (i.e. the files: **`sqlite-3_6_7.zip`** and **`sqlitedll-3_6_7.zip)`** from the location http://www.sqlite.org/download.html

Once you've unzipped the above two downloaded .zip files, copy the three extracted files (i.e. the files: **`sqlite3.def, sqlite3.dll`** and **`sqlite3.exe`**) to your Ruby bin directory (typically **`c:\ruby\bin`**).

Add **`sqlite3.dll`** and **`sqlite3.exe`** to your system environment variable - **`path`**.

We will use SQLite version 3.6.7

Note: Database driver allowing DataMapper to connect to SQLite

The Merb framework is ORM agnostic, which means that you can use Merb with ActiveRecord, DataMapper, and Sequel. The Merb core team recommends DataMapper.

Your initial Merb installation will have already installed DataMapper.

DataMapper uses the DataObjects drivers -
http://rubyforge.org/projects/dorb/
to connect to the database, so you will need to install **`do_mysql,`** or **`do_postgres`**. Drivers for Oracle and MS SQL are in the works. **`do_sqlite3`** for connecting to sqlite3 with DataMapper is bundled with merb.

Note: *Takaaki Kato* has suggested the following SQLite Manager addon for your Firefox browser –
https://addons.mozilla.org/en-US/firefox/addon/5817

This helps you manage any SQLite database on your computer.

Mongrel Installation

Installing Mongrel (http://mongrel.rubyforge.org/ - a fast HTTP library and server for Ruby that is intended for hosting Ruby web applications of any kind using plain HTTP) is quite simple. Type:

```
c:\> gem install mongrel
Successfully installed gem_plugin-0.2.3
Successfully installed cgi_multipart_eof_fix-2.5.0
Successfully installed mongrel-1.1.5-x86-mswin32-60
3 gems installed
Installing ri documentation for gem_plugin-0.2.3...
Installing ri documentation for cgi_multipart_eof_fix-2.5.0...
Installing ri documentation for mongrel-1.1.5-x86-mswin32-60...
Installing RDoc documentation for gem_plugin-0.2.3...
Installing RDoc documentation for cgi_multipart_eof_fix-2.5.0...
Installing RDoc documentation for mongrel-1.1.5-x86-mswin32-60...
c:\>
```

You can confirm whether mongrel is installed successfully, by typing:

```
c:\> gem specification mongrel
```

you will see all the details of the “mongrel” gem.

Note: Merb is “server-agnostic” (agnostic means independent). merb-core is built on rack (http://rack.rubyforge.org/doc/) and the rack adapter to use to run merb is by default mongrel. Merb supports emongrel, thin, ebb, fastcgi and webrick.

Just launch the application with:

```
merb -a whateverserver
```

That's it.

A Trivial Application

Merb comes with a set of code generators that will build parts of your application for you.

Generating code with merb-gen:

The easiest way to setup a new Merb application is to use **`merb-gen`**. **`merb-gen`** is available as part of the Merb bundle. **`merb-gen`** can produce different layouts.

```
merb-gen very_flat very_flat_app
```

Very flat Merb applications are the way to go when keeping everything in one file. You may find them useful for quick and dirty prototyping, micro-applications as well as proof-of-concept work. Unlike an HTML page you get server side code to make it possible to create little mini apps.

```
merb-gen flat flat_app
```

If you want to use view templates, but still prefer a single file for the bulk of your code, a flat layout may just be what you're looking for. This makes the flat layout optimal for simple Merb applications that do not use models but still have templates. It's also fairly easy to grow a flat layout into a standard one.

```
merb-gen app my_app
```

The standard layout provides the most comfort for most developers through a highly conventional and intelligent structure. This is recommended for nearly every application you develop.

Note that the Merb convention is that files are named in lowercase with underscores.

ruby_hello app

We shall develop a trivial application using the **very_flat** option. On a Windows XP box, open a command window and type:

```
C:\> merb-gen very_flat ruby_hello
Generating with very_flat generator:
←[32m     [ADDED]←[0m  ruby_hello.rb
←[32m     [ADDED]←[0m  Rakefile
←[32m     [ADDED]←[0m  spec
←[32m     [ADDED]←[0m  tasks/merb.thor
←[32m     [ADDED]←[0m  .gitignore
C:\>
```

This will produce a directory **ruby_hello** containing a single Ruby file along with some other useful files.

Note: **very_flat** apps use the app name as the controller, so you shouldn't use hyphens in them. Hyphens are not valid in class names.

Before we do anything else, let's run our application by using the **-I** flag to specify the init file for our Merb application. Change the folder to **ruby_hello** :

```
C:\> cd ruby_hello
```

And launch the application:

```
C:\ruby_hello> merb -I ruby_hello.rb
Loading init file from ruby_hello.rb
merb : worker (port 4000) ~ Starting Mongrel at port 4000
merb : worker (port 4000) ~ Successfully bound to port 4000
```
and you'll see the app boot up.

Now let's point a web browser to http://localhost:4000/, you should see the message returned by the value of the index action.

To stop the Merb server, press Control-C as shown:
```
^C
Terminate batch job (Y/N)? y
C:\ruby_hello>
```

Note: For more details on the various merb flags, type:

```
c:\> merb –H
c:\ruby_hello>
```

Close the command window.

A Simple Application

We shall now build a small Ruby Bookshop application and explore the various things we can do with Merb.

Generate a directory structure

To generate a directory structure for this new simple merb app (**ruby_simple_app**), on Windows, open a command window and change directory to say c:\ and type:

```
c:\> merb-gen app ruby_simple_app
```

Merb generates an application skeleton for you under directory **c:/ruby_simple_app**, including controllers, helpers, views, configuration, autotest configuration, and basic asset folders. Inside your app, you can use merb-gen to generate code for controllers, models, parts, and resources.

Change the folder to **ruby_simple_app** and type **dir** (on Windows) to see the generated directory structure:

```
c:\ruby_simple_app> dir
 Volume in drive c is Talim
 Volume Serial Number is 9066-B9C2
 Directory of c:\ruby_simple_app
12/14/2008  11:49 AM    <DIR>          .
12/14/2008  11:49 AM    <DIR>          ..
12/14/2008  11:49 AM               171 .gitignore
12/14/2008  11:49 AM    <DIR>          app
12/14/2008  11:49 AM    <DIR>          autotest
12/14/2008  11:49 AM    <DIR>          config
12/14/2008  11:49 AM    <DIR>          doc
12/14/2008  11:49 AM    <DIR>          gems
12/14/2008  11:49 AM    <DIR>          merb
12/14/2008  11:49 AM    <DIR>          public
12/14/2008  11:49 AM             1,136 Rakefile
12/14/2008  11:49 AM    <DIR>          spec
12/14/2008  11:49 AM    <DIR>          tasks
               2 File(s)          1,307 bytes
              11 Dir(s)  10,047,832,064 bytes free
c:\ruby_simple_app>
```

Remember to change version numbers in *your app's* **`ruby_simple_app/config/dependencies.rb`** to:

```
merb_gems_version = "1.0.7.1"
dm_gems_version   = "0.9.9"
```

We will be spending most of our time inside of the **app** directory. It's subdivided into models, views, controllers, and helpers.

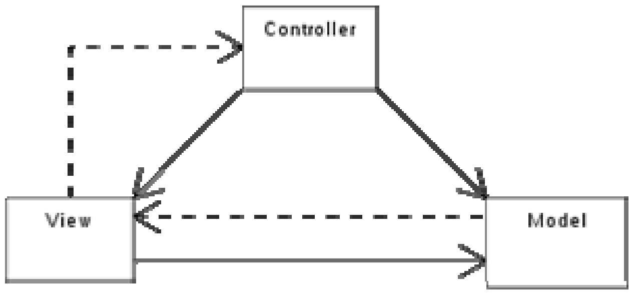
Models contain the business logic for your application, and its connection to a data store, like a database.

Views contain HTML or other templates.

Controllers are what tie the two together and handle actual requests from web browsers.

The goal is to keep as little logic as possible in the views, to make it easy to hand off your templates to a designer and to keep your application as maintainable as possible.

An easy way to understand MVC: the model is the data, the view is the window on the screen, and the controller is the glue between the two.
http://wiki.merbivore.com/howto/mvc



(**Image courtesy**: WikiMedia.org)

The **public** directory is meant to hold assets for your application to serve. These may include images, stylesheets, javascripts, and other static files.

In the **`app/views/layout`** folder is the generic application layout, which provides a standard "shell", including the doctype and info. The body of this page yields ("catch_content") to the various page contents, whether those be "static" or dynamic (dynamic for us in our example, later on, is "books" and "orders").

Finally, the **config** directory will hold various configuration details for your application. That's where we will store our database configuration.

init.rb

There is also an **`init.rb`** that contains all of the configuration details for our application itself. Let's take a look at that file.

The first line in a generated **`c:/ruby_simple_app/config/init.rb`** file is:

```
require 'config/dependencies.rb'
```

This loads the dependencies file, which lists the dependencies for your app. Merb's dependencies are just simple plugins. Merb will defer loading dependencies declared this way until the framework is fully loaded, to ensure that you're protected from load-order issues.

After loading in the dependencies, **`init.rb`** has some lines where by default, Merb uses DataMapper, RSpec, and ERB, as shown here:

```
use_orm :datamapper
use_test :rspec
use_template_engine :erb
```

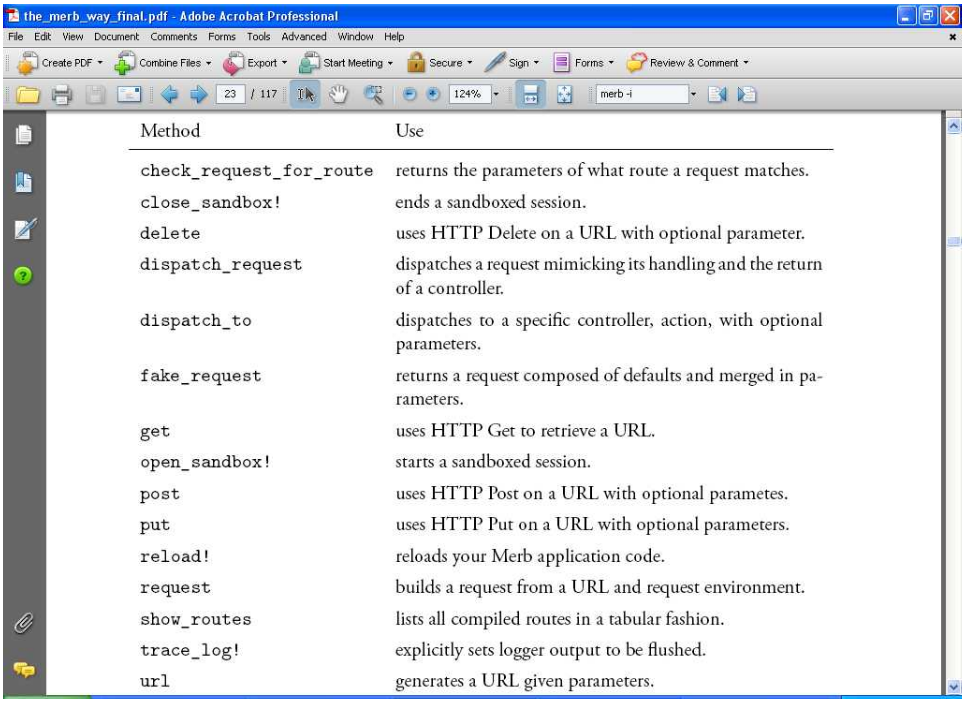
Interactive Merb

Interactive Merb allows you to do this by giving you an IRB session on a running Merb application. To do this, type:

```
c:\ruby_simple_app> merb -i
Loading init file from c:/ruby_simple_app/config/init.rb
Loading c:/ruby_simple_app/config/environments/development.rb
 ~ Connecting to database...
 ~ Loaded slice 'MerbAuthSlicePassword' ...
 ~ Compiling routes...
 ~ Activating slice 'MerbAuthSlicePassword' ...
irb: warn: can't alias context from irb_context.
irb(main):001:0>
```

This creates an exclusive interactive instance of your Merb application not bound to any port.

Now type **exit** to return to the command prompt.


the_merb_way_final.pdf - Adobe Acrobat Professional
File Edit View Document Comments Forms Tools Advanced Window Help
Create PDF Combine Files Export Start Meeting Secure Sign Forms Review & Comment
23 / 117
124%
merb -i

Method	Use
check_request_for_route	returns the parameters of what route a request matches.
close_sandbox!	ends a sandboxed session.
delete	uses HTTP Delete on a URL with optional parameter.
dispatch_request	dispatches a request mimicking its handling and the return of a controller.
dispatch_to	dispatches to a specific controller, action, with optional parameters.
fake_request	returns a request composed of defaults and merged in parameters.
get	uses HTTP Get to retrieve a URL.
open_sandbox!	starts a sandboxed session.
post	uses HTTP Post on a URL with optional parametes.
put	uses HTTP Put on a URL with optional parameters.
reload!	reloads your Merb application code.
request	builds a request from a URL and request environment.
show_routes	lists all compiled routes in a tabular fashion.
trace_log!	explicitly sets logger output to be flushed.
url	generates a URL given parameters.

(Image thanks to Foy Savas, author of the The Merb Way.)

The image above shows the various console methods available.

Building our first controller

All Merb controllers inherit from the class **`Merb::Controller`**. This controller fully implements the web functionality for controllers.

After first creating your Merb application with **`merb-gen`** you will find two files in **`app/controllers/`**. The first of these, Application (**`application.rb`**), will likely serve as the class from which your application's controllers subclasses. Note that application controller serves no other purpose but to unite the controllers of your application with a common superclass, and that without modification it is exactly **`Merb::Controller`**.

The exceptions controller (**`exceptions.rb`**) is subclassed from **`Merb::Controller`** and is used to create the error responses displayed to browsers.

Now let's get back to our simple application. We'll start by putting up a front-page for our website to welcome new visitors, and tell them something about the bookshop. Let's start by generating a controller for our front page.

Type:
```
C:\ruby_simple_app> merb-gen controller Static
Loading init file from c:/ruby_simple_app/config/init.rb
…
c:\ruby_simple_app>
```

Notice that Merb controllers don't have "Controller" after their name. The controller for pages is called *Pages*, and the controller for static stuff is called *Static*. *Static* is for pages that are not likely to change much and that won't require much dynamic content pulled from the database (e.g. the "home" page, "about us", "contact", etc.).

Open the file **app/controllers/static.rb** and take a look at the **index()** method. Inside controllers, methods that get called via the web are also called actions, and that's the terminology we'll use from now on. The index action is what is typically used for the default action within a controller, and that's how our application is setup by default. The return value of the action is what is eventually sent to the browser. We just have a call to the method **render** within **index**.

Let’s run our application and take a look at it. Start the Merb server by typing:

```
c:\ruby_simple_app> merb
Loading init file from c:/ruby_simple_app/config/init.rb
Loading c:/ruby_simple_app/config/environments/development.rb
 ~ Connecting to database...
 ~ Loaded slice 'MerbAuthSlicePassword' ...
 ~ Compiling routes...
 ~ Activating slice 'MerbAuthSlicePassword' ...
merb : worker (port 4000) ~ Starting Mongrel at port 4000
merb : worker (port 4000) ~ Successfully bound to port 4000
```

Now let’s point a web browser to http://localhost:4000/static and take a look. You will see the text “*You're in index of the Static controller.*” in a browser window. This has come from the *view* ie the file **/app/views/static/index.html.erb**

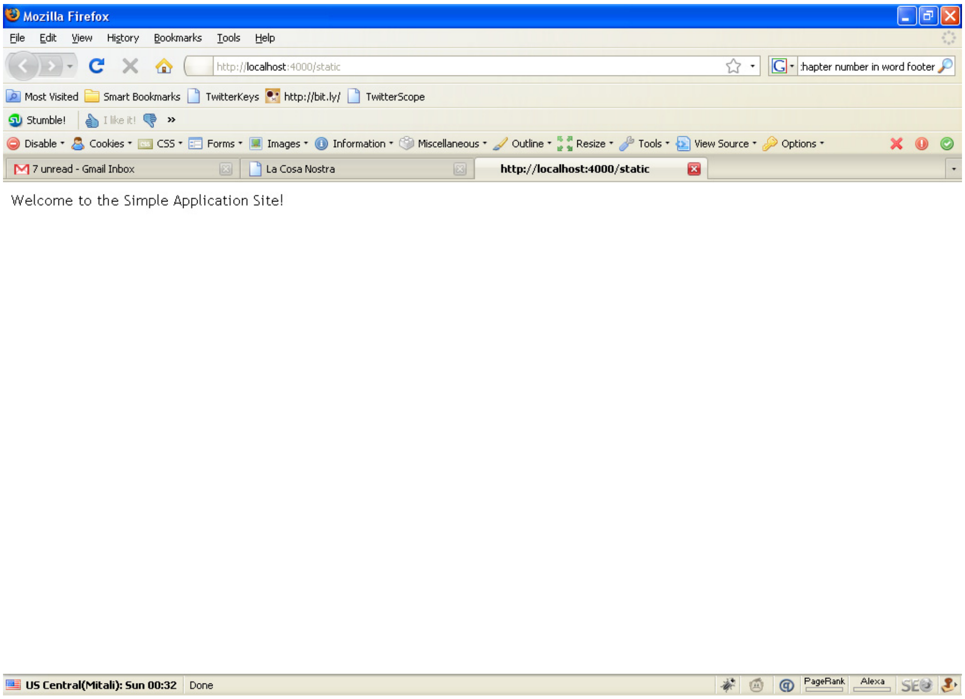
Now let’s add some code to the Static controller and make sure everything is working. Open the file **app/controllers/static.rb** and change **index()** to return *‘Welcome to the Simple Application Site!’* instead of calling **render**.

The modified static.rb file:

```
class Static < Application
# ...and remember, everything returned from an action
# goes to the client...
  def index
    'Welcome to the Simple Application Site!'
  end
end
```

Our Merb server is already running. Now let's point a web browser to `http://localhost:4000/static` and take a look.

Here's how it will look in the browser:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:4000/static
Most Visited Smart Bookmarks TwitterKeys http://bit.ly/ TwitterScope
Stumble! I like it!
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize Tools View Source Options
7 unread - Gmail Inbox
La Cosa Nostra
http://localhost:4000/static
US Central(Mitali): Sun 00:32 Done
PageRank Alexa

Welcome to the Simple Application Site!

To see a better resolution image click here – http://rubylearning.com/images/merb/fig1.png

Next, comment out the index method in **static.rb** file and add the following **hello** method in this controller –

```
class Static < Application
=begin
  def index
    #render
    "Welcome to the Ruby Bookshop Site!"
  end
=end
  def hello
    'Hello Static Merb!'
  end
end
```

Point your web browser to http://localhost:4000/static/hello and take a look. You should see the string '*Hello Static Merb!*' in the web browser window.

The **/static/hello** in the URL gets mapped to the *Static* controller and **hello** method. We also call the methods *actions*. The return value of the action is returned to the browser. In the **hello** action, it's the string '*Hello Static Merb!*'. Let's create another action named **time**.

```
def time
  "The time now is #{Time.now}."
end
```

The return value of the **time** action will be displayed when you go to http://localhost:4000/static/time.

Most of the time, you want to display a lot of data. And most of the time, you want to display data in html. You use *views* for this. Let's rewrite our **hello** action.

```
def hello
  render :action => 'hello'
end
```

Create **hello.html.erb** in the **app/views/static** directory. **hello** is the name of the view, *html* is the format, and *erb* is the template language used. If you're starting to learn merb, your views would most likely end with *.html.erb*. In **hello.html.erb**, write ***Hello Static Merb!***.

Now go to http://localhost:4000/static/hello and you should see the view.

Since the name of the action (**hello**) is the same as name of the view (**hello**), we can use **render** instead of **render :action => 'hello'**. The **hello** action can be written as:

```
def hello
  render
end
```

You can use html tags on your *views*. For example, to add a link, use a href tag. To make the development faster, Merb provides ruby methods which you can use in the views. These are called *helpers*. To add a link, you can use the **link_to** helper.

The view **app/views/static/hello.html.erb** can look like:

```
Hello Merb from the views. Click <%= link_to 'here',
'/static/time' %> to see the time.
```

The code above will be translated to:

```
Hello Merb from the views. Click <a href="/static/time">here</a>
to see the time
```

Confirm by pointing your browser to:
http://localhost:4000/static/hello

So far, we loaded our application in a browser by typing:
http://localhost:4000/static

It would be nice if we could load our application by typing:
http://localhost:4000/

Using a Router

Luckily, the Router handles the mapping of incoming URLs to controllers and actions (i.e. a router will guide incoming request to the code meant to respond to it.). By default, Merb loads router configuration from **`config/router.rb`**, so let's edit that file and add a default route.

```
Merb::Router.prepare do
 # add this line after default_routes
 match('/').to(:controller => 'static', :action => 'index')
end
```

First, a call to **`match()`** with the URL pattern. In this case, we want to match just the front page, so we use **`"/"`**. Next, we specify which controller and action will handle this route using **`to()`**.

Remember: To un-comment out the **`index`** method in **`static.rb`** file and point your browser to http://localhost:4000/ you should see the welcome message.

Note for Mac OS X Tiger (If you are using a version of Merb < to 1.0.7.1):
Re-starting Merb implies:

- close the browser window where http://localhost:4000/whatever is loaded
- Ctrl_C in terminal to kill merb
- start merb again

If step 1 is not done, then step 3 issues an error, port 4000 already in use:

```
merb : worker (port 4000) ~ FATAL: Could not bind to 4000. It was
already in use
```

In this case, close the browser window, then issue in the Terminal:

```
ps -auxww | grep merb
```

This will give you two lines with merb:

```
whatever   638  0.1  1,9   54848  19504  p1  S    8:54   10:49.65 merb : worker (port 4000)
whatever   701  0.0  0,0   27416    392  p1  R+   9:48    0:00.01 grep merb
```

The first one is the merb process, the second one is the grep merb you just launched.

On each line, you'll have first the process owner, then the job number, thereafter various flags, and at the end the name of the process. To kill the merb process, issue in the Terminal:

```
kill -9 638
```

This will kill the process number 638, which is the merb process.

Also, sometimes when starting merb, you may get the following error:

```
merb : worker (port 4000) ~ Starting Mongrel at port 4000
merb : worker (port 4000) ~
merb : worker (port 4000) ~ FATAL: Could not bind to 4000. It was
already in use
merb : worker (port 4000) ~
```

Using: merb -K 4000 also may not help. In such a situation, clear the problem by using a new terminal session.

The above will be corrected in merb 1.1
(The above note thanks to **Michele Garoche** and **George Thompson**).

Using Erubis

Merb supports the **Erubis** template language.
http://www.kuwata-lab.com/erubis/users-guide.html

Erubis is a fast, secure, and very extensible implementation of eRuby. eRuby means "embedded Ruby" in documents. Also, Erubis is implemented in Ruby. Erubis allows you to weave Ruby code into HTML pages. In fact, eRuby lets you use the entire power of Ruby to output HTML. An online chapter “HTML Templating with Erubis” is available here –
http://books.google.co.in/books?id=QC0rgGTmeNAC&pg=PA81&lpg=PA81&dq=HTML+Templating+with+erubis+apress&source=web&ots=aMKu3S20Hy&sig=9UAkmEZIfmpIn90hNy8aP-Xrp3o&hl=en&sa=X&oi=book_result&resnum=1&ct=result#PPA81,M1

The important things to know is that **<%= ruby_code %>** evaluates the ruby code and outputs the result, and **<% ruby_code %>** evaluates the code, but doesn't output anything.

In the default framework configuration, Merb looks for templates in a subdirectory of **app/views/** named for the controller. Since we're in the Static controller, we need to create our file in **app/views/static/**. The name of our new template should be **index.html.erb**.

The generators have already created this file for us, so let's replace the contents with some information about the Ruby book.

app/views/static/index.html.erb

```
<h1>Ruby eBook</h1>
<h3>The Ultimate Guide to Ruby Programming</h3>
<p class="float-left"><a
href="http://rubylearning.com/download/downloads.html"
title="Ruby eBook - US$ 9.95 only"><img
src="http://rubylearning.com/images/eBook5.jpg" alt="Ruby
eBook" /></a></p>
<p>Already <a
href="http://rubylearning.com/other/testimonials.html">over
20000</a> would-be Ruby programmers have downloaded the
Ruby eBook from <a
href="http://rubylearning.com/download/downloads.html"
>here</a>.</p>
```

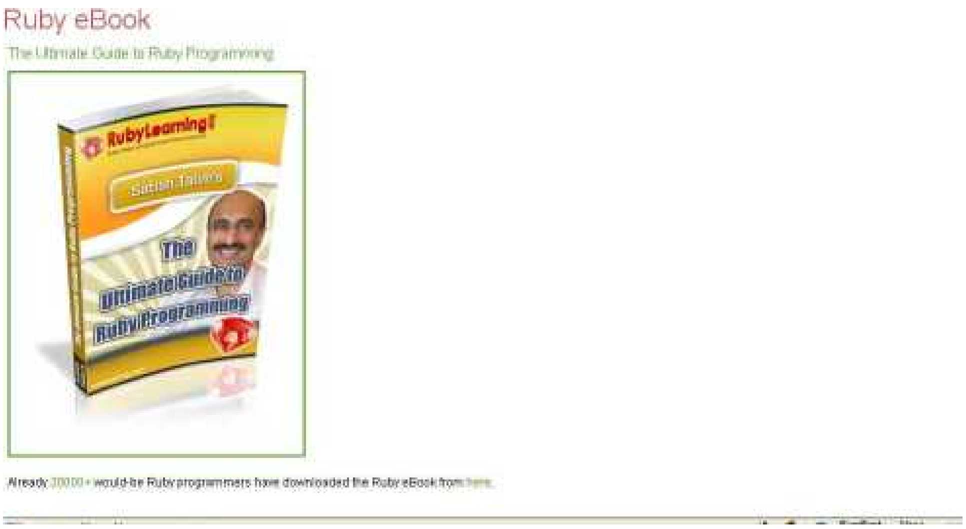
Also, let's modify our controller:
app/controllers/static.rb and add the **`render`** call in **`index`**

```
class Static < Application

  # ...and remember, everything returned from an action
  # goes to the client...
  def index
    render
    #'Welcome to the Simple Application Site!'
  end
  def hello
    #'Hello Static Merb!'
    render
  end
  def time
    "The time now is #{Time.now}."
  end
end
```

Point your browser to http://localhost:4000/

Here's how it looks:



Ruby eBook

The Ultimate Guide to Ruby Programming

Already 30000+ would-be Ruby programmers have downloaded the Ruby eBook from here.

Notice that Merb even styled it with a default set of CSS rules.

To see a better resolution image click here – http://rubylearning.com/images/merb/fig2.png

View Helpers

Helpers ("view helpers") are modules that provide methods which are automatically usable in your view. If you find yourself putting a lot of code in a view, think of writing a helper to handle that code, where it can be better managed and reused. They provide shortcuts to commonly used display code and a way for you to keep the programming out of your views. The purpose of a helper is to simplify the view. It is best if the view file is short and sweet, so you can see the structure of the output.

The folder **/app/helpers/** contains two helper files namely **global_helpers.rb** (contains methods available to all views of your application) and **static_helper.rb** (contains methods available to all views related to the static controller).

Let's add a **current_time** method to the helper **static_helper.rb**

```
module Merb
  module StaticHelper
    def current_time
      Time.now
    end
  end
end # Merb
```

Next, use this method in the view – **/app/views/static/index.html.erb** as:

```
<p>Already <a
href="http://rubylearning.com/other/testimonials.html">2000
0+</a> would-be Ruby programmers have downloaded the Ruby
eBook from <a
href="http://rubylearning.com/download/downloads.html">here
</a>.</p>
```

```
The current time is: <%= current_time %>
```

Point your browser to http://localhost:4000/ to see the time.

To stop the Merb server, press Control-C and close the command window.

Exercise 1

The above screen has a title – "Fresh Merb App". Search the **`c:\ruby_simple_app>`** folder for the file to edit, such that the title in the above screen is changed to "**Ruby Book Shop**".

Exercise 2

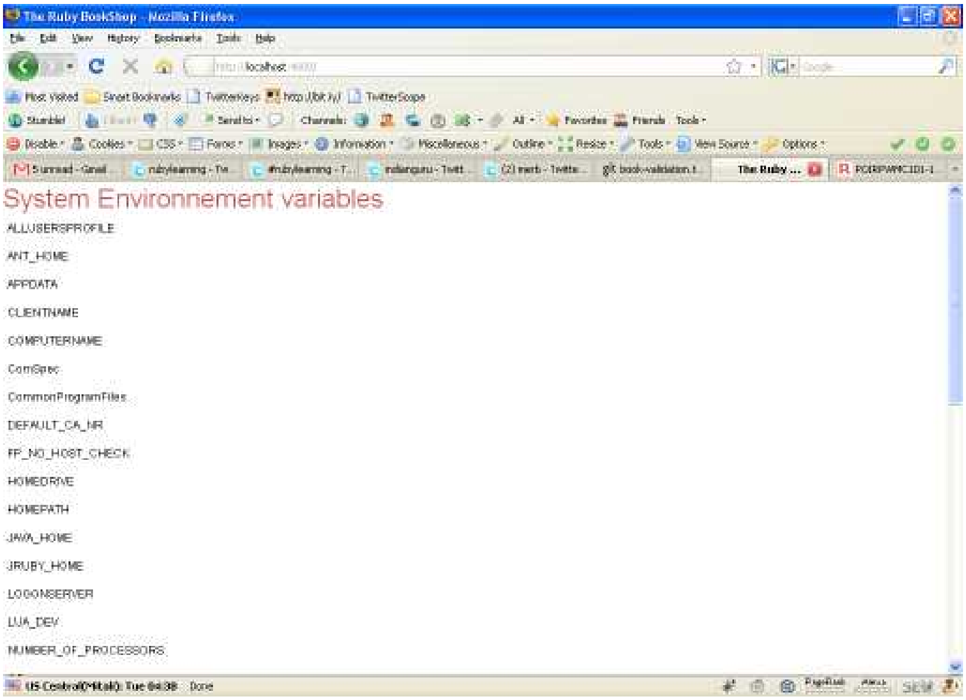
In this exercise, the following line should be printed at the end, in the screen being displayed in your browser now:

```
This site generated at: Sun Nov 23 11:46:28 +0530 2008
```

Note: The date and time will be your current date and time.
Hint: You should find another way than the one shown above for **`static_helper.rb`** to achieve the exercise.

Exercise 3

Write a new **`index.html.erb`** file and use Erubis code and Ruby **`ENV`** object to display something like this on your browser window:



System Environnement variables

ALLUSERSPROFILE

ANT_HOME

APPDATA

CLIENTNAME

COMPUTERNAME

ComSpec

CommonProgramFiles

DEFAULT_CA_NR

FP_NO_HOST_CHECK

HOMEDRIVE

HOMEPATH

JAVA_HOME

JRUBY_HOME

LOGONSERVER

LUA_DEV

NUMBER_OF_PROCESSORS

To see a better resolution image click here – http://rubylearning.com/images/merb/fig3.png

Typo: The header in the above screenshot should be System Environment variables (and not Environnement).

The Ruby Bookshop Application

Generate a new Merb app

We shall create a simple Ruby Bookshop application that has books, orders and authors. To generate a new Merb application with support for DataMapper (Ruby Bookshop app and directory - **ruby_bookshop**), on Windows change directory to say c:\ and type:

```
c:\> merb-gen app ruby_bookshop
```

Merb generates an application skeleton for you under directory **c:/ruby_bookshop**

Remember to change version numbers in *your app's* **ruby_bookshop/config/dependencies.rb** to:

```
merb_gems_version = "1.0.7.1"
dm_gems_version   = "0.9.9"
```

Merb Convention

Merb follows the convention wherein it maps database tables to classes. *If a database has a table called orders, our program will have a class named Order (under app/models).* Rows in this table correspond to objects of the class - a particular order is represented as an object of class Order. Within that object, attributes are used to get and set the individual columns. Our Order object has methods to get and set the amount, the sales tax, and so on.

Therefore an ORM layer maps tables to classes, rows to objects, and columns to attributes of those objects. Class methods are used to perform table-level operations, and instance methods perform operations on the individual rows.

Also:

- database table names, like variable names, have lowercase letters and underscores between the words
- **table names are always** *plural*
- files are named in lowercase with underscores

Creating a Resource: Book

A Merb Resource is simply a way to interact with a database model over HTTP. We can generate a model, controller, and a set of default views all with one command by using the resource generator.

You can read what all properties are allowed for a model here – http://wiki.merbivore.com/howto/models#properties

and

http://merbivore.com/documentation/1.1.0/doc/rdoc/stack/index.html?a=C00000176&name=Property

Let's build a resource for our **books** table:
`books → name, price, description, id`

Change folder to **`c:\>ruby_bookshop`** and type:

```
c:\ruby_bookshop> merb-gen resource book
name:string,description:text,price:float
Loading init file from c:/ruby_bookshop/config/init.rb
…
c:\ruby_bookshop>
```

The first argument, **`resource`** is the name of the generator to call. The second argument, **book**, is the name of our new resource. **It will create a model called book and a controller called books, in keeping with the convention.** The rest of the arguments are fields to store in our new resource that describes the name of the field, and the type of the field. For instance, the database table will have **name** column, which will be a string.

Note: Please do not use spaces between attributes, when generating a resource. Therefore do not type as shown below:

```
merb-gen resource book
name:string,[space]description:text,[space]price:big_decimal
```

Let's take a quick look at the model that Merb generated for us.

File: **C:/ruby_bookshop/app/models/book.rb**

```
class Book
  include DataMapper::Resource
  property :id, Serial
  property :name, String
  property :price, Float
  property :description, Text
end
```

Merb created a stub model, with the properties set up as we described in the generator.

Generating another Model: Order

The purpose of an ORM (DataMapper in our case) is to hide the fact that you're working with a database, and to make code accessing your database seamlessly integrate with your other Ruby code.

For each book, we'll probably want to store the name of the book, a description, and a price. We might also want another database table (Orders) where we could store a record every time someone purchases a book. Because book prices might change, we'll probably want to store both a link to the book purchased, as well as the price at time of purchase in that record. So we can have:

orders → price, customer_name, id

In order to create the order model, we could use the model generator:

```
c:\ruby_bookshop> merb-gen model order
customer_name:string,price:float
Loading init file from c:/ruby_bookshop/config/init.rb
…
c:\ruby_bookshop>
```

Setting Relationships

Note that we already generated the book model when we generated the book resource. Information about associations is in the MERB API documentation:
http://merbivore.com/documentation/1.1.0/doc/rdoc/stack/index.html?a=M000133&name=has#

Now, one book can have n orders. We'll need to add in the **has n** and **belongs_to** associations. Open **book.rb** and **order.rb** and add those in, as shown next:

```
class Book
  include DataMapper::Resource
  property :id, Serial
  property :name, String
  property :price, Float
  property :description, Text
  has n, :orders
end

class Order
  include DataMapper::Resource
  property :id, Serial
  property :customer_name, String
  property :price, Float
  belongs_to :book
end
```

Make changes to Model Book

(This note thanks to *Michele Garoche*)
File to change: **c:/ruby_bookshop/app/models/book.rb**

Add validations

Reference:
http://wiki.merbivore.com/howto/models#options

We have already declared that the type of book price is **`Float.`**

Now, we will specify the scale and precision for price.

```
property :price, Float, :scale => 2, :precision => 5
```

Now, we will provide our own error message when the price is not in the range [-999.99, 999.99], though the message is not very accurate (not a number).

```
property :price, Float, :scale => 2, :precision => 5,
 :message => "Please, enter a price in the range [0.01, 999.99]."
```

Also, we will specify the maximum length for the Book's name:

```
property :name, String, :within => 3..40
```

This will check that the book's name has at least 3 characters and at most 40 characters, but does not check that the characters are significant.

Now, impede null entries:

```
property :price, Float, :nullable => false, :precision => 5,
:scale => 2,
  :message => "Please, enter a price in the range [0.01, 999.99]."
property :name, String, :nullable => false, :within => 3..40
```

This has the advantage for the price to put 0.0 in the price field when returning from validation.

Now, provide default entry for Description:

```
property :description, Text, :default => "To be provided"
```

This way, some text will appear even when the description has not been entered.

Adding built-in validations

We will add them at the end of the file.

```
validates_is_unique :name
```

It will check that the name entered is unique, i.e. does not exist already in the database. Notice that this is not sufficient to ensure a real visual uniqueness, as you may enter a name with spaces at the end without being given any message error.

Adding custom validations

This is made in two steps:

Declare the method used to do the validation

```
validates_with_method :price, :positive_price?
validates_with_method :name, :sensible_name?
```

The first symbol is the symbol matching the property we want to validate. The second symbol may be named whatever you want. It returns true when successful, or an array composed of false and the appropriate error message. As the method returns true or false, it is best to give it a question mark at the end.

Write effectively the methods

Within the method, the access to a property is made via **`self.property`**:

```
def positive_price?
  if self.price <= 0.0
    [false, "Please, enter a positive price."]
  else
    true
  end
end
```

This method checks that the price is not negative or null. Coupled with the built-in validations, it will impede to create a book with a price of $0.00.

```
def sensible_name?
  stripped_name = self.name.strip
  if stripped_name == '' || stripped_name.length < 3
    [false,  "Please,  enter  a  sensible  name  with  at  least  3 
significative characters."]
  else
    true
  end
end
```

This method first strips the name from spaces, and then checks that the stripped name has at least 3 characters.

To summarize, here is the model (**book.rb**):

```
  class Book
    include DataMapper::Resource

    property :id, Serial

    property :price, Float, :nullable => false, :precision => 5,
:scale => 2, :message => "Please, enter a price in the range
[0.01, 999.99]."
    property :name, String, :nullable => false, :within => 3..40
    property :description, Text, :default => "To be provided"

    has n, :orders

    validates_is_unique :name
    validates_with_method :price, :positive_price?
    validates_with_method :name, :sensible_name?

    def positive_price?
      if self.price <= 0.0
        [false, "Please, enter a positive price."]
      else
        true
      end
    end

    def sensible_name?
      stripped_name = self.name.strip
      if stripped_name == '' || stripped_name.length < 3
        [false, "Please, enter a sensible name with at least 3
significative characters."]
      else
        true
      end
    end
  end
```

Make some changes to the controller

File: **c:/ruby_bookshop/app/controllers/books.rb**

This is not strictly necessary, but it will help understand which message is rendered when. We will add the name of the book created or modified in the redirect resource part.

In the create method:

```
redirect resource(@book), :message => {:notice => "Book
#{@book.name} was successfully created."}
```

In the update method:

```
redirect resource(@book), :message => {:notice => "Book
#{@book.name} was successfully updated."}
```

Note that the message here is a notice message, not an error message. This is an important distinction which we will use later when we change the css stylesheet and the layout for the application.

Here are the two methods in **books.rb** after the change:

```
  def create(book)
    @book = Book.new(book)
    if @book.save
      redirect resource(@book), :message => {:notice => "Book
#{@book.name} was successfully created."}
    else
      message[:error] = "Book failed to be created"
      render :new
    end
  end

  def update(id, book)
    @book = Book.get(id)
    raise NotFound unless @book
    if @book.update_attributes(book)
      redirect resource(@book), :message => {:notice => "Book
#{@book.name} was successfully updated."}
    else
      display @book, :edit
    end
  end
```

Push those definitions into the database

Simply run:

```
c:\ruby_bookshop> rake db:automigrate
(in c:/ruby_bookshop)
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/rake.rb
 ~ Connecting to database...
 ~ Loaded slice 'MerbAuthSlicePassword' ...
 ~ Compiling routes...
 ~ Activating slice 'MerbAuthSlicePassword' ...
c:\ruby_bookshop>
```

This will create the **books** and **orders** table for you to use in your application.

Note: Automigrations (**db:automigrate**) work by completely **trashing any existing table and rebuilding it from scratch**. This works great for development, where you don’t have any mission-critical data in your database.

Use the database name from database.yml

Let's take a look how the file **config/database.yml** helps DataMapper connect. The YAML file is broken down into three sections, one for each environment. Here, **adapter** specifies the type of database program you're running (and incidentally the **adapter** DataMapper will have to use to communicate with it). The database name is **sample_development.db**.

Check the Schema

If you launch sqlite3 and take a look at the schema, you'll see that the database is all set up and ready to go. Type:

```
c:\ruby_bookshop> sqlite3 sample_development.db
SQLite version 3.6.7
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE "books" ("id" INTEGER NOT NULL PRIMARY KEY
AUTOINCREMENT, "price" FLOAT(5,2) NOT NULL, "name"
VARCHAR(50) NOT NULL, "description" TEXT DEFAULT 'To
be provided');
CREATE TABLE "orders" ("id" INTEGER NOT NULL PRIMARY KEY
AUTOINCREMENT, "customer_name" VARCHAR(50), "price" FLOAT,
"book_id" INTEGER);
CREATE TABLE "sessions" ("session_id" VARCHAR(32) NOT NULL,
"data" TEXT DEFAULT 'BAh7AA== ', "created_at" DATETIME,
PRIMARY KEY("session_id"));
CREATE TABLE "users" ("id" INTEGER NOT NULL PRIMARY KEY
AUTOINCREMENT, "login" VARCHAR(50), "crypted_password"
VARCHAR(50), "salt" VARCHAR(50));
sqlite> .quit
c:\ruby_bookshop>
```

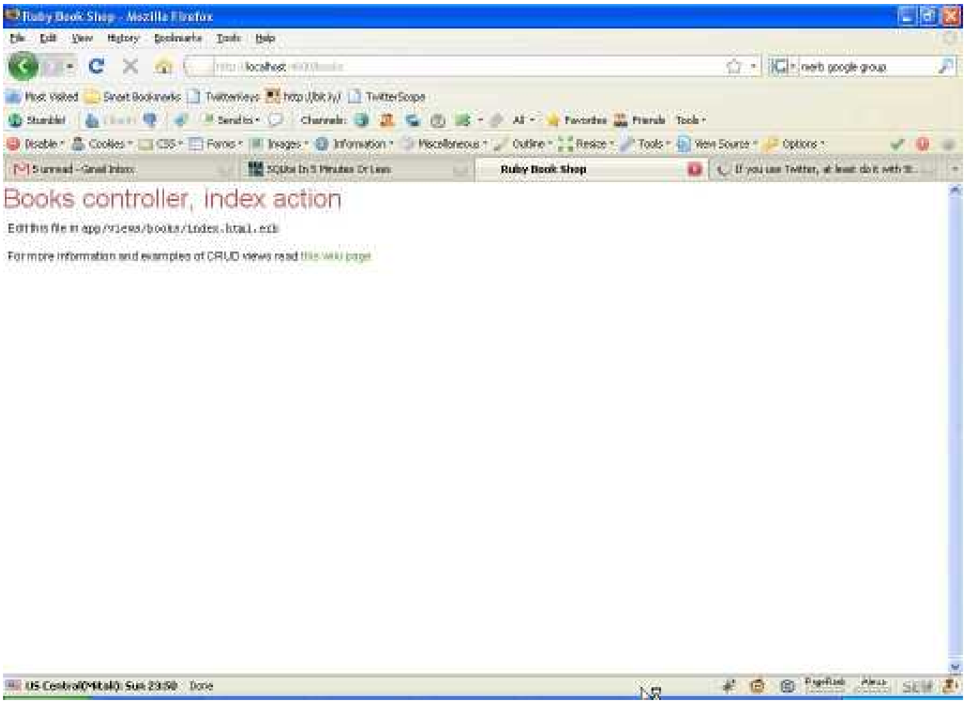
Merb and DataMapper took care of syncing up your code with your databases. It also created a couple of tables for sessions and users, which come with the merb stack. **Note**: This is not strictly correct. Observe that we put in name :within => 3..40, but in the schema, there is VARCHAR(50), which is the length by default for string. It would have not happened if we would have used migration. In this case, we would have had VARCHAR(40) for name.

Open a new console window and change the folder to **`c:\ruby_bookshop`** and type:

```
c:\ruby_bookshop> merb
```

Next, point your browser to http://localhost:4000/books. You will see the default page that just provides some basic details about where you are in the app.

The screen you see has a title – "Fresh Merb App". Edit **`c:/ruby_bookshop/app/views/layout/application.html.erb`** and change the title to "**Ruby Book Shop**", as shown below:


Ruby Book Shop - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:4000/books
Ruby Book Shop

Books controller, index action

Edit this file in app/views/books/index.html.erb

For more information and examples of CRUD views read this wiki page

To see a better resolution image click here – http://rubylearning.com/images/merb/fig4.png

The purpose of some generated files

`c:/ruby_bookshop/app/views/books/index.html.erb`
This is the view that will show a list of books. **Note** that the above page (see image above) is being generated by this file.

c:/ruby_bookshop/app/views/books/edit.html.erb
This is the view that will allow you to edit a book.

c:/ruby_bookshop/app/views/books/show.html.erb
This is the view that will show you a single book.

c:/ruby_bookshop/app/views/books/new.html.erb
This is the view that will allow you to create a new book.

c:/ruby_bookshop/app/controllers/books.rb
The books controller

Another thing that the generator did for us is to create a route for the resource. Merb's router allows you to specify, in Ruby, where in our application to send inbound requests. By default, Merb loads router configuration from **config/router.rb** which has:

```
Merb::Router.prepare do
  resources :books
  …
  default_routes
end
```

For now, it's enough to recognize that Merb has provided our users with a way to get to every action in the Books controller.

Callable tasks

One important thing which may help you to know how to call the different available tasks, is the following:

```
C:\ruby_bookshop> rake --tasks
(in c:/ruby_bookshop)
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
rake audit:actions              # Print out contr...
rake audit:controllers          # Print out all c...
rake audit:routes               # Print out the n...
…
```

As you can see, it gives the name of all callable tasks under rake with an explanation. Due to the number of outdated tutorials (as Merb evolves much more rapidly than the tutorials), it saves some hours/days of searching the exact syntax to launch a task.

Insert some records into the Books table

Remember:
Controller -> books.rb -> class Books
Model -> book.rb -> class Book
Table -> books

Let us insert some records in our books table by using the merb console. In the command window type:

```
c:\ruby_bookshop> merb -i
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
 ~ Connecting to database...
 ~ Loaded slice 'MerbAuthSlicePassword' ...
 ~ Compiling routes...
 ~ Activating slice 'MerbAuthSlicePassword' ...
irb: warn: can't alias context from irb_context.
irb(main):001:0> Book.create(:name => 'The Ultimate Guide to Merb
Programming', :description => 'The best Ruby guide to get you
started in Merb', :price => 3.0)
 ~ SELECT "id" FROM "books" WHERE ("name" = 'The Ultimate Guide to
Merb Programming') ORDER BY "id" LIMIT 1
 ~ INSERT INTO "books" ("description", "name", "price") VALUES
('The best Ruby guide to get you started in Merb', 'The Ultimate
Guide to Merb Programming', 3.0)
=> #<Book id=1 price=3.0 name="The Ultimate Guide to Merb
Programming" description="The best Ruby guide to get you started
in Merb">
irb(main):002:0> Book.create(:name => 'The Merb Way', :description
=> 'An upcoming book on Merb', :price => 4.0)
 ~ SELECT "id" FROM "books" WHERE ("name" = 'The Merb Way') ORDER
BY "id" LIMIT 1
 ~ INSERT INTO "books" ("description", "name", "price") VALUES
('An upcoming book on Merb', 'The Merb Way', 4.0)
=> #<Book id=2 price=4.0 name="The Merb Way" description="An
upcoming book on Merb">
irb(main):003:0> quit
c:\ruby_bookshop>
```

Create a helper for Views

We will need to add some parts for rendering the error and notice messages created before.

In **app/helpers** we will edit the generated **books_helper.rb** file which will contain the formatting for the price:

```
module Merb
  module BooksHelper
    def format_currency(num)
      format('$%0.2f', num)
    end
  end
end
```

In showing the price above, we used Ruby's built-in formatting tools, **f** stands for floating point with two decimal places. This will guarantee that 2, 2.0, and 2.00 will all display as "$2.00".

The day we decide to change the formatting we have only to change this helper and it will be applied in all the views.

List the Books

We know that file –
`c:/ruby_bookshop/app/views/books/index.html.erb`
is the view that will show a list of books and has the following contents - a generated header and text (to delete):

```
<h1>Books controller, index action</h1>
<p>Edit this file in
<tt>app/views/books/index.html.erb</tt></p>
<p>For more information and examples of CRUD views read <a
href="http://wiki.merbivore.com/howto/crud_view_example_wit
h_merb_using_erb"> this wiki page</a>
```

First, let's replace the text in <h1> with "Books to Buy". Next, we will delete the text in <p></p> and replace it with:

```
<h1>Books to Buy</h1>
<table>
  <thead>
    <tr>
      <td>Name</td>
      <td>Price</td>
    </tr>
  </thead>
  <tbody>
    <% @books.each do |book| %>
      <tr>
        <td><%= book.name %></td>
        <td><%= format_currency(book.price) %></td>
        <td><%= link_to 'Show', resource(book) %></td>
        <td><%= link_to 'Edit', resource(book, :edit) %></td>
        <td><%= delete_button book %></td>
      </tr>
    <% end %>
  </tbody>
</table>
<%= link_to 'New', resource(:books, :new) %>
```

Here we have called the function **`format_currency(num)`** created earlier, to format the price inside the views. We also use a new helper called **resource**. This helper allows you to go to a url through the routing map for a given resource, based upon how you set up the resource in your router (**config/router.rb**). In this case, **resource(book)** will generate **/books/<id>**. **resource(book, :edit)** will generate **/books/<id>/edit**, which the router will send to the edit action in the Books controller.

A way to see the routes is to use the merb console:

```
merb –i
Merb::Router::named_routes
Merb::Router::resource_routes
Merb::Router::routes
```

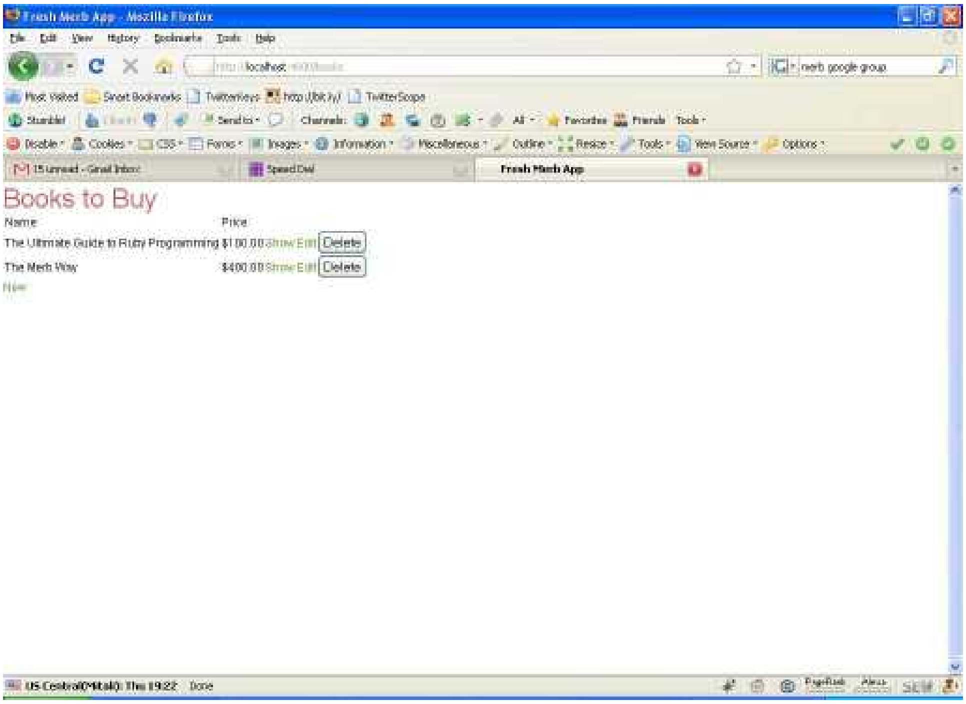
Notice how named_routes are related to an action, resource_routes are related to a pair model/controller and routes are related to a pair controller/action.

The **delete_button** method from **Merb::Helpers::Form** generates a HTML *functional* delete button –

Reference:
http://merbivore.com/documentation/1.0/doc/rdoc/merb-helpers-1.0/index.html?a=M001017&name=delete_button

The **index.html.erb** template would iterate over all of the available books. At the moment, all it does it provide links to show, edit, or delete the book. The *merb* server is already running, point your browser to:
http://localhost:4000/books

and here’s how it looks:



Books to Buy

Name	Price		
The Ultimate Guide to Ruby Programming	$100.00	Show Edit	Delete
The Merb Way	$400.00	Show Edit	Delete

New

To see a better resolution image click here – http://rubylearning.com/images/merb/fig5.png

Note: Please be aware that there is no confirmation dialog for deletion with the delete button.

Sandboxing

Interactive Merb is capable of Sandboxing your activity, allowing you to mess around with database resources only to have all changes reverted when you exit the sandbox.

Note: **As of now, the ORM DataMapper does not support this feature**. **As such, please do not use this feature.**

```
c:\ruby_bookshop> merb -i
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
```

```
 ~ Connecting to database...
 ~ Loaded slice 'MerbAuthSlicePassword' ...
 ~ Compiling routes...
 ~ Activating slice 'MerbAuthSlicePassword' ...
irb: warn: can't alias context from irb_context.
irb(main):001:0> Book.count
 ~ SELECT COUNT(*) FROM "books"
=> 3
irb(main):002:0> merb.open_sandbox!
Loading development environment in sandbox (Merb 1.0.7.1)
Any modifications you make will be rolled back on exit
=> [Merb::Orms::DataMapper]
irb(main):003:0> Book.all.destroy!
 ~ DELETE FROM "books"
=> true
irb(main):004:0> Book.count
 ~ SELECT COUNT(*) FROM "books"
=> 0
irb(main):005:0> merb.close_sandbox!
Modifications have been rolled back
=> nil
irb(main):006:0> Book.count
 ~ SELECT COUNT(*) FROM "books"
=> 0
irb(main):007:0> exit
c:\ruby_bookshop>
```

The following explanation on the merb console is worth bookmarking – **http://merb.4ninjas.org/#_crud**

Displaying a Book

On the index page, Merb provided a link to Show the book. That link pointed to **url(:book, book)**. This URL will point to the Books controller, and the **show** action.

Here's the default **show** (action) generated by Merb in **books.rb**:

```
def show(id)
  @book = Book.get(id)
  raise NotFound unless @book
  display @book
end
```

First, we get the book, based on the id that was passed in. If no book was found, we raise a **NotFound** error, which will be returned to the client as a 404 error. Finally, if a book was found, we display it.

The template for the show action is in **show.html.erb -**

```
<h1>Books controller, show action</h1>
<p>Edit this file in
<tt>app/views/books/show.html.erb</tt></p>
<p>For more information and examples of CRUD views read <a
href="http://wiki.merbivore.com/howto/crud_view_example_wit
h_merb_using_erb"> this wiki page</a>
```

We shall modify the same as in:

```
<h1>Book: <%= @book.name %></h1>
<p><%= @book.description %></p>
<p>Price: <%= format_currency(@book.price) %></p>
<%= link_to 'Edit', resource(@book, :edit) %> |
<%= link_to 'Back', resource(:books) %>
```

Note: **merb-helpers** is a plugin for the Merb web framework that provides different view helpers.
http://merbivore.com/documentation/1.0/doc/rdoc/merb-helpers-1.0/index.html

Creating a new Book

Here's the default **new** (action) generated by Merb in **books.rb**:

```
def new
  only_provides :html
  @book = Book.new
  display @book
end
```

The first line of the method tells Merb to override the default mime type support for the class and provide only HTML. The second line creates a new book, and the third line displays it.

The generated template for the **new** action is in **new.html.erb**:

```
<h1>Books controller, new action</h1>
<p>Edit this file in
<tt>app/views/books/new.html.erb</tt></p>
<p>For more information and examples of CRUD views read <a
href="http://wiki.merbivore.com/howto/crud_view_example_wit
h_merb_using_erb"> this wiki page</a>
```

As we did before, the first step is to change the <h1> to something more descriptive, and remove the placeholder <p>. We'll title this page Make a new Book. We'll want to replace the placeholder with an actual form, using Merb's form helpers.

```
<h1>Make a new Book</h1>
<%= form_for(@book, :action => url(:books) ) do %>
<p>Name of book: <%= text_field :name, :size =>"40" %></p>
<p><%= " " * 5 %>Description: <%= text_area :description, :cols =>
"40", :rows => "2", :wrap => "SOFT" %></p>
<p><%= " " * 17 %>Price: <%= text_field :price, :size => "7"
%></p>
<p><%= submit "Create" %></p>
<% end =%>
<%= link_to 'Back', url(:books) %>
```

Some details of **`form_for`** –
http://wiki.merbivore.com/howto/simple_edit_form

Partials

This allows us to extract out common bits of template code into a single template that is referenced from each template that uses it. Partials are stored in the same directory as the templates themselves, with a leading underscore. For instance, we would call the form partial **`_form.html.erb`**.

Partials documentation –
http://merbivore.com/documentation/1.1.0/doc/rdoc/stack/index.html?a=C00000233&name=RenderMixin

Now extracting out the common parts from the new template, the form partial would look like (in file **`_form.html.erb`**):

```
<p>Name of book: <%= text_field :name, :size =>"40" %></p>
<p><%= " " * 5 %>Description: <%= text_area :description, :cols =>
"40", :rows => "2", :wrap => "SOFT" %></p>
<p><%= " " * 17 %>Price: <%= text_field :price, :size => "7"
%></p>
<p><%= submit "Create" %></p>
```

The modified **new.html.erb** file is:

```
<h1>Make a new Book</h1>
<%= form_for(@book, :action => url(:books) ) do %>
  <%= partial :form %>
<% end =%>
<%= link_to 'Back', url(:books) %>
```

Editing a Book

Here's the default **edit** (action) generated by Merb in **books.rb**:

```
def edit(id)
  only_provides :html
  @book = Book.get(id)
  raise NotFound unless @book
  display @book
end
```

Like the **new** form, this action only provides HTML. Next, we grab the book matching the ID passed in. If no book was found, we raise a **NotFound** error, which is sent back to the browser as a 404. Otherwise, we render the template.

The generated template for the **edit** action (in file **edit.html.erb**) Is:

```
<h1>Books controller, edit action</h1>
<p>Edit this file in
<tt>app/views/books/edit.html.erb</tt></p>
<p>For more information and examples of CRUD views read <a
href="http://wiki.merbivore.com/howto/crud_view_example_wit
h_merb_using_erb"> this wiki page</a>
```

The new **edit.html.erb** file is:

```
<h1>Edit a Book</h1>
<%= form_for(@book, :action => resource(@book)) do %>
  <%= partial :form1 %>
<% end =%>
<%= link_to 'Back', url(:books) %>
```

The form for new action was called **_form.html.erb**. The form for edit action is called here **_form1.html.erb** and contains:

```
<p>Name of book: <%= text_field :name, :size =>"40" %></p>
<p><%= "&nbsp;" * 5 %>Description: <%= text_area :description,
    :cols => "40", :rows => "2", :wrap => "SOFT" %></p>
<p><%= "&nbsp;" * 17 %>Price: <%= text_field :price, :size => "7"
%></p>
<p><%= submit "Update" %></p>
```

Some DRY code

You notice that apart from the last line, the code in both **_form.html.erb** and **_form1.html.erb** is the same. Hence we will DRY it.

We will modify the **_form.html.erb** form, first to handle error messages and second to make it possible to use it with both new and edit views, as the variables printed are the same.

Put at the beginning of the file **app/views/books/_form.html.erb**:
<%= error_messages_for @books %>

It will display all the error messages which will have been gathered during the validation process.

Next, we will drop the last line, which is not common to edit and new forms. Therefore, let us edit **app/views/books/new.html.erb** and add:
<%= submit "Create" %>

The complete **app/views/books/new.html.erb** now is:

```
<h1>Make a new Book</h1>
<%= form_for(@book, :action => url(:books) ) do %>
  <%= partial :form %>
  <%= submit "Create" %>
<% end =%>
<%= link_to 'Back', url(:books) %>
```

The current **edit.html.erb** file is referring to **:form1**. Instead, we need to refer to **:form** (the new common form for edit and new views) instead of **:form1**. Also, add the line:
<%= submit "Update" %>

Hence the modified **edit.html.erb** is:

```
<h1>Edit a Book</h1>
<%= form_for(@book, :action => resource(@book)) do %>
  <%= partial :form %>
  <%= submit "Update" %>
<% end =%>
<%= link_to 'Back', url(:books) %>
```

Finally, drop the **app/views/books/_form1.html.erb**, which was the former form for edit action.

The final **app/views/books/_form.html.erb** is:

```
<%= error_messages_for @books %>
<p>Name of book: <%= text_field :name, :size =>"40" %></p>
<p><%= "&nbsp;" * 5 %>Description: <%= text_area :description,
    :cols => "40", :rows => "2", :wrap => "SOFT" %></p>
<p><%= "&nbsp;" * 17 %>Price: <%= text_field :price, :size => "7"
%></p>
```

(The above customization of the form is credited to *Warren Li*.)

Rendering

We should modify slightly, the rendering of notice and error messages to get them right, because they are supposed to be wrapped into a **<div class='notice'>** and a **<div class='error'>** respectively.

In **app/views/layout/application.html.erb**, change the body contents to:

```
<body>
  <div class='notice'>
    <%= message[:notice] %>
  </div>
  <div class='error'>
    <%= message[:error] %>
  </div>
  <%= catch_content :for_layout %>
</body>
```

catch_content(:for_layout) tells merb that you want the content for the action to render in this spot. **:for_layout** is the default content that each action renders.
http://merbivore.com/documentation/1.0.6.1/doc/rdoc/stack/index.html?a=C00000233&name=RenderMixin

Next, in **public/stylesheets/master.css** add some css styles to the stylesheet for messages.

For example, add the following at the end of the **public/stylesheets/master.css** file:

```
div.error, div.notice {
  margin: 0px 5px 0px 0px;
  padding: 5px;
  text-align: left;
}
div.error ul, div.notice ul {
  margin-left: 15px;
}
div.error {
  color: #c55;
}
div.notice {
  color: #4d9b12;
}
```

Our merb server is still running, point your browser to:
`http://localhost:4000/books`

Summary

1. Generate a Merb application with support for DataMapper

```
c:\> merb-gen app ruby_bookshop
```

2. Create a Resource for a **Book**

```
c:\ruby_bookshop> merb-gen resource book
name:string,description:text,price:float
```

This will create a model called book and a controller called books, in keeping with the convention.

3. Generating another Model: **Order**

```
c:\ruby_bookshop> merb-gen model order
customer_name:string, price:float
```

4. Setting Relationship between **Book** and **Order**
Add in the has n and belongs_to associations in **`book.rb`** and **`order.rb`**

5. Make changes to Model **Book** and add Validations
book.rb and **books.rb**

6. Make some changes to the controller in **books.rb create** and **update** methods

7. Push those definitions into the database

```
c:\ruby_bookshop> rake db:automigrate
```

8. Use the database name from **`database.yml`**
It is **`sample_development.db`**

9. Check the Schema

```
c:\ruby_bookshop> sqlite3 sample_development.db
sqlite> .schema
```

Remember:
Controller -> books.rb -> class Books
Model -> book.rb -> class Book
Table -> books

10. Insert some records into the **Books** table

```
c:\ruby_bookshop> merb -i
irb(main):001:0> Book.create(:name => 'The Ultimate Guide to Merb
Programming', :description => 'The best Ruby guide to get you
started in Merb', :price => 3.0)
```

11. Create a helper for Views

12. List the Books
The view to list out the books is -
`c:/ruby_bookshop/app/views/books/index.html.erb`

13. Display (Show) a Book
The view to show a book is -
`c:/ruby_bookshop/app/views/books/show.html.erb`

14. Create (New) a Book
The view to a new book is -
`c:/ruby_bookshop/app/views/books/new.html.erb`

Optional: Using Partials / DRY code / Rendering -
`c:/ruby_bookshop/app/views/books/_form.html.erb`

15. Edit a Book
The view to edit a book is -
`c:/ruby_bookshop/app/views/books/edit.html.erb`

16. Some DRY code

17. Rendering

Deletion

One possible drawback in merb is that one does not require to confirm, while deleting an object. See:

http://wiki.merbivore.com/howto/simple_delete_view

You can add a snippet to enable confirmation:

```
<td><%= delete_button book, 'Delete', :class => 'delete'%></td>
```

Add the following to **public/javascripts/application.js**

```
$(document).ready(function() {
 $('.delete').click(function() {
 var answer = confirm('Are you sure?');
 return answer;
 });
});
```

Don't forget to load jquery.js and application.js in **app/views/layout/application.html.erb** at the end of the <head> section.

```
<script src="/javascripts/jquery.js" type="text/javascript"
></script>
 <script src="/javascripts/application.js"
type="text/javascript"></script>
```

(The above note on delete, thanks to *Doug Sparling* and *Massimiliano Giroldi*).

Authors-Books association

(This note thanks to *George Thompson* and *Michele Garoche*)

Objective: We would like to:

- create, show, display authors as we did for books in the part authoring
- delete authors only if there are no book associated with them
- integrate authors data into books views

The model for authors will be the following:

- last_name, 3 to 40 significant characters
- first_name, 3 to 40 significant characters
- biography, a text field with default text

The validations on book will change to the following:

- instead of having uniqueness on the name, it will have uniqueness on the name + author_id when creating a new book

The validations on authors will be the following:

- same types of validations as for the book
- it will have uniqueness on first name + last name only when creating a new author

Change the route

At the end of the file **config/router.rb**, add:
match('/').to(:controller => 'books', :action =>'index')

Generate a *resource* for an authors table

This should generate a model, controller, and a set of default views.
authors → first_name, last_name, biography, id

Type:

```
c:\ruby_bookshop> merb-gen resource author
first_name:string,last_name:string,biography:text
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
Generating with resource generator:
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
Loading init file from c:/ruby_bookshop/config/init.rb
Loading E:/ruby_bookshop/config/environments/development.rb
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
←[32m     [ADDED]←[0m  spec/models/author_spec.rb
←[32m     [ADDED]←[0m  app/models/author.rb
←[32m     [ADDED]←[0m  spec/requests/authors_spec.rb
←[32m     [ADDED]←[0m  app/controllers/authors.rb
←[32m     [ADDED]←[0m  app/views/authors/index.html.erb
←[32m     [ADDED]←[0m  app/views/authors/show.html.erb
←[32m     [ADDED]←[0m  app/views/authors/edit.html.erb
←[32m     [ADDED]←[0m  app/views/authors/new.html.erb
←[32m     [ADDED]←[0m  app/helpers/authors_helper.rb
←[1;32mresources :authors route added to config/router.rb←[0m
```

```
c:\ruby_bookshop>
```

As we saw previously, Merb creates all the files we need to integrate Authors into our application. Merb even updates our routes by adding the authors resource. Shut down the previously working *merb* server by pressing **^C** and then restart the *merb* server. Let's try to view our new resource in the browser:
http://localhost:4000/authors

If everything went well you should see something like this:

```
Authors controller, index action.

Edit this file in app/views/authors/index.html.erb

For more information and examples of CRUD views read this wiki
page
```

Add association to Author and Book models

So far we have a Books model and an Authors model. *In our model we are going to assume that Books have one Author and that Authors can have many books (Yes, I know that a book can have more than one author. Please humor us for now).* Let Merb know about these new associations. **Hint**: Use **`belongs_to.`** Our **`book.rb`** is:

```
  class Book
    include DataMapper::Resource
    property :id, Serial
    property :price, Float, :nullable => false, :precision => 5, :scale => 2, :message => "Please, enter a price in the range [0.01, 999.99]."
    property :name, String, :nullable => false, :within => 3..40
    property :description, Text, :default => "To be provided"

    has n, :orders
    belongs_to :author

    validates_is_unique :name
    validates_with_method :price, :positive_price?
    validates_with_method :name, :sensible_name?

    def positive_price?
      if self.price <= 0.0
        [false, "Please, enter a positive price."]
      else
        true
      end
    end

    def sensible_name?
      stripped_name = self.name.strip
      if stripped_name == '' || stripped_name.length < 3
        [false, "Please, enter a sensible name with at least 3 significative characters."]
      else
        true
      end
    end
  end
```

Next, set up the one-to-many relationship between the Author and Books. **Hint**: Use **`has n.`** Our **`author.rb`** is:

```
class Author
  include DataMapper::Resource
  property :id, Serial
  property :biography, Text
  property :first_name, String
  property :last_name, String
  has n, :books
end
```

Add validations to the Author model

Here are the validations we want to do:
On first name and last name:

- it should not be null
- it should have between 3 and 40 characters
- it should not be composed of only spaces
- it should have uniqueness on last name + first name when creating a new author

On biography:

- it should provide a default text

The validations will look like this:

```
    property :last_name, String, :nullable => false, :within =>
3..40
    property :first_name, String, :nullable => false, :within =>
3..40
    property :biography, Text, :default => "To be provided"

    has n, :books

    validates_with_method :last_name, :sensible_last_name?
    validates_with_method :first_name, :sensible_first_name?
    validates_with_method :last_name, :name_unique?, :if =>
:new_record?

    def sensible_last_name?
      stripped_name = self.last_name.strip
      if stripped_name == '' || stripped_name.length < 3
        [false, "Please, enter a sensible name with at least 3
significant characters."]
      else
        true
      end
    end

    def sensible_first_name?
      stripped_name = self.first_name.strip
      if stripped_name == '' || stripped_name.length < 3
        [false, "Please, enter a sensible name with at least 3
significant characters."]
      else
        true
      end
    end

    def name_unique?
      fullname = full_name(self.first_name, self.last_name)
      if (self.class.all :first_name => self.first_name,
:last_name => self.last_name) != []
        [false, "The author #{fullname} exists already."]
      else
        true
      end
    end
```

The first two **validates_with_method** methods are equivalent to the one used in book model.

Notice the third **validates_with_method** has now a new key pair:
:if => :new_record?

It is what is called a conditional validation. It will be run only when we will create a new record. Failing to add this conditional test would have lead to an error message when trying to update a record.

The method **name_unique?** associated with the third **validates_with_methods** first joins the first and last name via a method (which will be written later on), then it searches if there is already an author with the given last name and given first name in the table. If it finds one, it prepares an error message, otherwise it allows the validation.

Adding properties and changing validations in the book model

We will have to integrate the author_id property to make a unique index on it and the book name. This way we will ensure that the combination of book name and author_id is unique. At the same time, we will suppress the validation on name's uniqueness.

In **book.rb**, the property part of the book model becomes:

```
    property :id, Serial

    property :price, Float, :nullable => false, :precision => 5,
:scale => 2,
            :message => "Please, enter a price in the range [0.01,
999.99]."
    property :name, String, :nullable => false, :within => 3..40
    property :description, Text, :default => "To be provided"
    property :author_id, Integer, :index => :unique
```

We comment out the line on name's uniqueness:
#validates_is_unique :name

Also, we add a new validation for uniqueness on book name + author_id:
```
validates_with_method :name, :name_authorid_unique?, :if =>
:new_record?
```

This method will be:

```
    def name_authorid_unique?
      nameauthor = [self.name, self.author_id].join(' ')
      if (self.class.all :name => self.name, :author_id =>
self.author_id) != []
          [false, "The book #{name} exists already for the author
" + full_name( self.author.first_name, self.author.last_name) +
"."]
        else
          true
        end
      end
```

As already explained for the author model, the **`validates_with_method`** method introduces a customized method and a conditional, to make sure to validate it only when we create a new book. This method joins the book's name and the author_id and checks if there is already a book with this name and this author_id in the table. Then it prepares an error message if it finds one, otherwise it allows the validation.

Drying the code in author and book models

Notice that we use the same function in both author and book models to check for a name with at least 3 significant characters, the only thing which is different is the field on which it applies.

We are going to wrap this common part in a method, and put it in a module accessible to both models and views.

Note that this cannot go into **`app/helpers`**, since the helpers are only accessible to the views, not to the models. Instead we shall put it in the **`ruby_bookshop/gems`** folder which can be loaded from the **`config/dependencies.rb`** folder.

We will also write the full_name method, which concatenates the first name and last name of the author in this helper. This method will be put in **`global_helpers`** as we need it for all views, and also in the authorsbooks helper as we need it in the models.

Writing the helper

We will name this helper **`authorsbooks_helper.rb`** and put it in the **`ruby_bookshop/gems`** folder. The code is just:

```
module AuthorsBooksHelper
  def name_not_empty?(stripped_name)
    if stripped_name == '' || stripped_name.length < 3
      [false, "Please, enter a sensible name with at least 3 
significative characters."]
    else
      true
    end
  end

  def full_name(str1, str2)
    str1 + " " + str2
  end

end
```

Writing the full_name method in global_helpers

We will change here the file **`app/helpers/global_helpers.rb`**. It will become:

```
module Merb
  module GlobalHelpers
    # helpers defined here available to all views.
    def full_name(str1, str2)
      str1 + " " + str2
    end
  end
end
```

What all this means is that global_helpers are accessible to all views, but not to controllers or models. Modules inside gems folder and loaded from config/dependencies are accessible to all models and controllers, but not to views.

Change the models

Now we will change the models to use the above helper. This means including the helper into the models and use this helper inside the **`sensible_name?`** function, in both models.

To include the helper, just put:
`include AuthorsBooksHelper`

in both models at the beginning just after the first include. Notice the convention for the name, capitalising each plural parts and parts separated by underline:
`authorsbooks_helper` becomes **`AuthorsBooksHelper`**

Now we change the **`sensible_name?`** method in **`book.rb`** to:

```
    def sensible_name?
      stripped_name = self.name.strip
      name_not_empty?(stripped_name)
    end
```

You'll see that inside the model we have access to any field with **`self`**.

Now we change the **`sensible_last_name?`** and **`sensible_first_name?`** method in **`author.rb`** to:

```
    def sensible_last_name?
      stripped_name = self.last_name.strip
      name_not_empty?(stripped_name)
    end

    def sensible_first_name?
      stripped_name = self.first_name.strip
      name_not_empty?(stripped_name)
    end
```

The updated **book.rb** file is:

```
class Book
  include DataMapper::Resource
  include AuthorsBooksHelper

  property :id, Serial

  property :price, Float, :nullable => false, :precision => 5,
:scale => 2,
          :message => "Please, enter a price in the range [0.01,
999.99]."
  property :name, String, :nullable => false, :within => 3..40
  property :description, Text, :default => "To be provided"
  property :author_id, Integer, :index => :unique

  has n, :orders
  belongs_to :author

  #validates_is_unique :name
  validates_with_method :price, :positive_price?
  validates_with_method :name, :sensible_name?
  validates_with_method :name, :name_authorid_unique?, :if =>
:new_record?
  def positive_price?
    if self.price <= 0.0
      [false, "Please, enter a positive price."]
    else
      true
    end
  end

  def sensible_name?
    stripped_name = self.name.strip
    name_not_empty?(stripped_name)
  end

  def name_authorid_unique?
    nameauthor = [self.name, self.author_id].join(' ')
    if (self.class.all :name => self.name, :author_id =>
self.author_id) != []
        [false, "The book #{name} exists already for the author "
+ full_name( self.author.first_name, self.author.last_name) + "."]
      else
        true
      end
    end
end
```

The updated **author.rb** file is:

```
class Author
  include DataMapper::Resource
  include AuthorsBooksHelper

  property :id, Serial

  property :last_name, String, :nullable => false, :within =>
3..40
  property :first_name, String, :nullable => false, :within =>
3..40
  property :biography, Text, :default => "To be provided"

  has n, :books

  validates_with_method :last_name, :sensible_last_name?
  validates_with_method :first_name, :sensible_first_name?
  validates_with_method :last_name, :name_unique?, :if =>
:new_record?

  def sensible_last_name?
    stripped_name = self.last_name.strip
    name_not_empty?(stripped_name)
  end

  def sensible_first_name?
    stripped_name = self.first_name.strip
    name_not_empty?(stripped_name)
  end

  def name_unique?
    fullname = full_name(self.first_name, self.last_name)
    if (self.class.all :first_name => self.first_name, :last_name
=> self.last_name) != []
      [false, "The author #{fullname} exists already."]
    else
      true
    end
  end
end
```

Loading the helper at server launch time

To load it, we will put the following line at the end of the **`config/dependencies.rb`** file:

```
require File.join(Merb.root, "/gems/",
"authorsbooks_helper.rb")
```

This loads the file named **`authorsbooks_helper.rb`** located in the **`gems`** folder situated just under the root of the Merb application.

Migration

DataMapper provides two strategies: autoupgrade and classic migrations. Automigrations work by completely trashing any existing table and rebuilding it from scratch. This works great for development, where you don't have any mission-critical data in your database. For production, though, alarm bells are probably going off in your head.

The first alternate strategy is called autoupgrade, which will attempt to modify the existing schema to match your models, but will never destroy data. For instance, if you add a new column, it will add it to the database without deleting your data. Unfortunately, autoupgrade has its drawbacks. For one, it can't handle column renaming, or, at the moment, changes to the properties of a column (for instance, adding a default to a column is not yet supported). For robust control over live, production data, we're going to need a different strategy – classic migration.

We use classic migrations that will load data for our models - users, books, and now authors. The good thing about setting the migration is that we can run them as many times as we need and not have to manually reenter our data.

We have first to create data for authors first, as data for books will contain authors' references.

Create Authors table

We have already seen how to create a table inside a merb console. This time we will do it with migration, just to see how it can be done:

```
c:\ruby_bookshop> merb-gen migration create_authors_table
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
Generating with migration generator:
←[32m     [ADDED]←[0m
schema/migrations/001_create_authors_table_migration.rb
c:\ruby_bookshop>
```

It will generate a file in **c:/ruby_bookshop/schema/migrations** folder which will be named: **001_create_authors_table_migration.rb**.

The name of the file generated is as follows:
xxx-nameofthemigrationfile_migration.rb

where xxx is a number automatically added. Here it is 001 since it is the first migration file generated.

We will populate it with the following lines:

```
migration 1, :create_authors_table  do
  up do
    create_table :authors do
      column :id, Integer, :serial => true
      column :last_name, String, :nullable => false, :size => 40
      column :first_name, String, :nullable => false, :size => 40
      column :biography, DataMapper::Types::Text, :default => "To
be provided"
    end
  end

  down do
    drop_table :authors
  end
end
```

What we have done here in the **up** part of the migration is merely copy the properties lines from the authors model, and make some changes in it:

Change property into column in all lines:
to define the columns

Add Integer type and change Serial into :serial => true in id column:
the first change is necessary to give a type to the column, the second change makes the column a primary key auto-incremented

Change :within => 3..40 in to :size => 40 in last_name and first_name columns:
as a table has no notion of interval for the length, we change it to a size and gives it the maximum wanted size

Change Text into DataMapper::Types::Text in biography column:
Text is not a recognised Ruby type, hence we change it to a DataMapper type.

Notice that the **down** part is only given as a way of writing for database engines which supports drop table. This is not the case of SQLite when there are foreign constraints in a table, which is the case here with the association between Author and Book models. If you need to rollback the migration for some reason, see “Troubleshooting the migrations”, later on in this eBook.

Generate a migration file for authors

The name of the migration file is up to you, but it is better to give it a sensible name. Here it is named **`insert_data_in_authors`**. Type:

```
c:\ruby_bookshop> merb-gen migration insert_data_in_authors
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
Generating with migration generator:
←[32m     [ADDED]←[0m
schema/migrations/002_insert_data_in_authors_migration.rb
c:\ruby_bookshop>
```

It will generate a **002_insert_data_in_authors_migration.rb** file inside **schema/migrations** folder.

Edit the migration file to add the **up** and **down** methods. The **up** migration consists here in inserting two rows in the table.

```
migration 2, :insert_data_in_authors  do
      up do
        Author.create({ :id => 1,
        :first_name => 'Satish',
        :last_name => 'Talim',
        :biography => 'To be provided'})

        Author.create({ :id => 2,
        :first_name => 'Foy',
        :last_name => 'Savas',
        :biography => 'To be provided'})
      end

      down do
        Author.get(1).destroy
        Author.get(2).destroy
      end
end
```

The **up** part uses the create method which is a shortcut to define and save. The **down** part deletes only the data which has been created by this migration. To do it, it gets the row by id and destroys it.

Thus: the **up** part is to effectively run the migration, the **down** part to rollback it.

Generate migration for books

The migration for books implies first to add a column to the table, the **`author_id`** column.

We will use this command to create the migration:

```
c:\ruby_bookshop> merb-gen migration add_columnauthorid_in_books
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
Generating with migration generator:
←[32m     [ADDED]←[0m
schema/migrations/003_add_columnauthorid_in_books_migration.rb
c:\ruby_bookshop>
```

It will generate a file in **`c:/ruby_bookshop/schema/migrations`** folder which will be named: **`003_add_columnauthorid_in_books_migration.rb`**.

In the **`up`** part we need first to add the column author_id, then to update this column for the data already in the table.

The code will be:

```
    migration 3, :add_columnauthorid_in_books  do
      up do
        modify_table :books do
          add_column("author_id", Integer)
        end
        book = Book.get(1)
        book.author_id = 1
        book.save

        book = Book.get(2)
        book.author_id = 2
        book.save
      end


    down do
        modify_table :books do
          drop_column(author_id)
        end
      end
    end
```

Notice how the addition of the column is made in a loop, this ensures that the column exists when we modify it thereafter.

The modification of the data in author_id column is based on retrieving the book by id and then inserting in the author_id column the right data.

The **down** part consists in dropping the author_id column. This is only given as a way to do it for database engines which support dropping a column. *This is not the case of SQLite*.

Run the migration

As we add a new table and a column to another table, we need to run the migrations up to the last. We will do it with:

```
c:\ruby_bookshop> rake db:migrate:up[3]
```

It will do all the migrations from the last one done not included it to the fourth one.

Add views for authors

Here, we want to impede deletion of an author if a book exists for it. To do this, we will modify the model to trigger a function just before the destroy method, then in the controller we will show an error message when returning.

Change in author model

The change in author model is:

```
before :destroy do
  throw :halt if books.count > 0
end
```

This code will generate a bad answer from the server if there is a book linked to the author. Notice that since there is an association between the authors and the books table, we have access to the books from the author model.

Change in authors controller

The current code for destroy action in the authors controller is the following:

```
    def destroy(id)
      @author = Author.get(id)
      raise NotFound unless @author
      if @author.destroy
        redirect resource(:authors)
      else
        raise InternalServerError
      end
    end
```

With the change made previously in the model, the controller will raise an internal server error when we ask for deleting an author which has a book associated to it.

Instead of raising an **`InternalServerError`** in this case, we will redirect the resource as if there were a delete action and give it an error message.

The code will look like this:

```
    def delete(id)
      @author = Author.get(id)
      raise NotFound unless @author
      if @author.destroy
        redirect resource(:authors),
        :message => {:notice => "Author " +
full_name(@author.first_name,@author.last_name) + " was
successfully deleted"}
      else
        redirect resource(:authors),
        :message => {:error => "You cannot delete author " +
full_name(@author.first_name,@author.last_name) + "; it has
books."}
      end
    end
```

Notice that we have added the author first and last name to the message to make it more significant.

Modify views for books

We have to take into account the author data into our views.

What we want to have here is a select drop-down list composed of the first name and last name of the author to show and edit the data.

Changes for new and show actions

We need only the change the
`c:\ruby_bookshop\app\views\books\_form_html.erb`
file for new and show actions.

The code will look like this:

```
    <%= error_messages_for @books %>
    <p>Name of book: <%= text_field :name, :size =>"40" %></p>
    <p><%= "&nbsp;" * 13 %>Author: <%= select :author_id,
:collection => Author.all.map {|u| [u.id, full_name(u.first_name,
u.last_name)]} %></p>
    <p><%= "&nbsp;" * 5 %>Description: <%= text_area :description,
:cols => "40", :rows => "2", :wrap => "SOFT" %></p>
    <p><%= "&nbsp;" * 17 %>Price: <%= text_field :price, :size =>
"7" %></p>
```

The collection symbol allows us to declare a list, which will be composed of the authors' first and last names.

Changes for index action

Here we will simply use our full_name function to display the author's first and last name and add a header for author in the table with some css code which we will add later to the stylesheet.

The code will look like this:

```
<h1>Books to Buy</h1>
    <table class="liste">
      <tr>
        <th>Book's name</th>
        <th>Author</th>
        <th>Price</th>
        <th colspan="3">Action</th>
      </tr>
      <% @books.each do |book| %>
      <tr>
        <td><%= book.name %></td>
        <td><%= full_name(book.author.first_name, 
book.author.last_name) %></td>
        <td class="right"><%= format_currency(book.price) %></td>
        <td class="small"><%= link_to 'Show', resource(book) 
%></td>
        <td class="small"><%= link_to 'Edit', resource(book, 
:edit) %></td>
<td class="small"><%= delete_button book, 'Delete', :class => 
'delete' %></td>
      </tr>
      <% end %>
    </table>
<%= link_to 'New', resource(:books, :new) %>
```

Modifying authors views

We will make the same kind of changes to authors views as we did for books views

Changes for index action

As with book index view, we will have a table with a header for author's full name.

The new **`index.html.erb`** fie is the following:

```
<h1>Authors List</h1>
<table class="liste">
 <tr>
  <th>Name</th>
  <th colspan="3">Action</th>
 </tr>
 <% @authors.each do |author| %>
 <tr>
  <td><%= full_name(author.first_name,
author.last_name) %></td>
  <td class="small"><%= link_to 'Show',
resource(author) %></td>
  <td class="small"><%= link_to 'Edit',
resource(author, :edit) %></td>
<td class="small"><%= delete_button author, 'Delete', :class =>
'delete' %></td>
 </tr>
 <% end %>
</table>
<%= link_to 'New', resource(:authors, :new) %>
```

Changes for new action

Similar to the corresponding view for books, the new **`new.html.erb`** file becomes:

```
<h1>Create a new Author</h1>
<%= form_for(@author, :action => url(:authors) ) do %>
  <%= partial :form %>
  <%= submit "Create" %>
<% end =%>
<%= link_to 'Back', url(:authors) %>
```

Changes for show action

Again similar to the corresponding view for books, the new **show.html.erb** file is:

```
<h1>Name: <%= full_name(@author.first_name, @author.last_name)
%></h1>
<p><%= @author.biography %></p>
<%= link_to 'Edit', resource(@author, :edit) %> | <%= link_to
'Back', resource(:authors) %>
```

Changes for edit action

Item the new **edit.html.erb** file is:

```
<h1>Edit an Author</h1>
<%= form_for(@author, :action => resource(@author)) do %>
  <%= partial :form %>
  <%= submit "Update" %>
<% end =%>
<%= link_to 'Back', url(:authors) %>
```

Write the _form.html.erb file

Very similar to the corresponding book form, it is:

```
<%= error_messages_for @authors %>
<p>First name: <%= text_field :first_name, :size =>"40" %></p>
<p>Last name: <%= text_field :last_name, :size =>"40" %></p>
<p>Biography: <%= text_area :biography, :cols => "40", :rows =>
"2", :wrap => "SOFT" %></p>
```

Modify the stylesheet

As it is, the stylesheet is not very appealing, here is a proposal which improves it a bit:

Change the general margin from 0 to 1 near the top of the file

```
* {
  margin: 1px;
  padding: 0px;
  text-decoration: none;
}
```

Remove the background-color on a and a:hover

```
a {
  color: #4d9b12;
  /* background-color: #fff; */
  text-decoration: none;
}
a:hover {
  color: #4d9b12;
  /* background-color: #fff; */
  text-decoration: underline;
}
```

Add the necessary to handle the tables just before the div.error, div.notice near the end of the file

```
table.liste {
      margin-top: 10px;
      margin-bottom: 10px;
      border-width: 2px;
      border-spacing: 0px;
      border-style: outset;
      border-color: white;
      border-collapse: separate;
      background-color: #ffffef;
}
table.liste th {
      border-width: 1px;
      padding: 2px 10px 3px 10px;
      border-bottom: outset 2px white;
    font-size: 12px;
}
table.liste td {
      border-width: 1px;
      padding: 5px 10px 3px 10px;
      border-style: none;
      color: black;
      background-color: #ffffef;
}
table.liste td.right {
      text-align: right;
}
table.liste td.small {
      font-size: 9px;
      padding: 5px;
}
```

Testing the application

Now we could add the following authors and books via the views in the application. Modify them, delete them, try to add the same book twice, try to delete an author which has a book.

```
    book name: Devin's Little Book of Jokes
    author name: Devin Eastham
    description: Jokes for people who don't laugh
    price: 8.95

    book name: The Adventures of Tito
    author name: Tito The Dog
    description: Tito is always getting into trouble. Follow his
hilarious coming of age story
    price: 3.95

    name: Devin's Cookbook
    author name: George Thompson
    description: Food that kids can make
    price: 2.99
```

Troubleshooting the migrations

In this exercise we have done migrations where rollback is not supported by SQLite:

- adding a column is supported, but not dropping a column

- adding a table is supported, but not dropping a table which has foreign constraints in it.

Consequently, if you need for some reasons to migrate down one of these migrations, you have to do it directly in the database.

First we enter the database:

```
    sqlite3 sample_development.db
```

Next, at the prompt, we enter the .tables command we give us the list of all tables:

```
  sqlite> .tables
    authors          migration_info  sessions
    books            orders          users
```

The migration_info table receives the list of all the migration already performed successfully. To see them, enter at the prompt the **`select * from migration_info;`** command:

```
    sqlite> select * from migration_info;
    add_user_in_users
    insert_data_in_books
    create_authors_table
    insert_data_in_authors
    add_columnauthorid_in_books
```

This table has a schema file where you can see the name of the column. To see the schema, enter at the prompt the **`.schema migration_info`** command:

```
    sqlite> .schema migration_info
    CREATE TABLE "migration_info" ("migration_name"
VARCHAR(255) UNIQUE);
```

Now, say you have a problem with the authors table, to get rid of it, you need to:

a - delete all the migrations lines down to and included the first one dealing with authors in the migration_info table

You will do it with those commands:

```
    sqlite> delete from migration_info where migration_name
= "create_authors_table";
    sqlite> delete from migration_info where migration_name
= "insert_data_in_authors";
    sqlite> delete from migration_info where migration_name
= "add_columnauthorid_in_books";
```

b - delete the authors table

```
    drop table authors;
```

c - quit the sqlite3 tool:

```
     .quit
```

d - make the necessary change in the files if needed

If the problem is in a model file or migration file, change it, and save it. If the problem is in the data, no changes are needed.

e - rerun the migration up

As previously, you will run the migration up again with:

```
       db:migrate:up[3]
```

MerbAuth – The Basics

Support for user authentication is part of the Merb framework. MerbAuth is the authentication framework for your application. With MerbAuth it’s possible to consider a user to be any kind of ruby object. It could be an instance of class User, Person, Account, Staff, Student, Customer, Employee, String, Url, FileIO, Hash, Array.

MerbAuth has a couple of parts. **merb-auth-core**, **merb-auth-more**, and **merb-auth-slice-password**. We will look at **merb-auth-core**.

When we created our application **ruby_bookshop**, the authentication was already configured:

```
#c:/ruby_bookshop/app/models/user.rb
# This is a default user class used to activate merb-auth.
# You will need to setup your database and create a user.
class User
  include DataMapper::Resource

  property :id,     Serial
  property :login,  String
end
```

and files:

```
c:/ruby_bookshop/merb/merb-auth/setup.rb
c:/ruby_bookshop/merb/merb-auth/ strategies.rb
```

We need to add a user first, as follows:

```
c:\ruby_bookshop> merb -i
Loading init file from c:/ruby_bookshop/config/init.rb
Loading c:/ruby_bookshop/config/environments/development.rb
 ~ Connecting to database...
 ~ Loaded slice 'MerbAuthSlicePassword' ...
 ~ Compiling routes...
 ~ Activating slice 'MerbAuthSlicePassword' ...
irb: warn: can't alias context from irb_context.
irb(main):001:0> u = User.new
=> #<User id=nil login=nil crypted_password=nil salt=nil>
irb(main):002:0> u.login = 'satish'
=> "satish"
irb(main):003:0> u.password = u.password_confirmation = 'talim'
=> "talim"
irb(main):004:0> u.save
 ~ INSERT INTO "users" ("crypted_password", "login", "salt")
VALUES ('a32418013e
23614a10abf780f1d2059f5f84ab0d', 'satish',
'03f2f8ee0cab59f4e55dc32d892e88669293
6276')
=> true
irb(main):005:0> exit
c:\ruby_bookshop>
```

Notice that we should add **`password_confirmation`** and equals it to the password to honour the requirements on password. Actually, under the hood, merb will transform the password into an encrypted password.

Next, add authentication to the book controller for **`edit`** and **`new`** action. In practice, we will exclude from authentication the **`show`** and **`index`** methods of our application.

In **`app/controllers/books.rb`** add:
`before :ensure_authenticated, :exclude => [:show, :index]` as shown below:

```
class Books < Application
  # provides :xml, :yaml, :js
  before :ensure_authenticated, :exclude => [:show, :index]

  def index
...
```

Now try and edit a book. Launch merb, point your browser to http://localhost:4000/books and click on the Edit button. You should be given a login, password window, where you have to enter the login and password you've just created above. Once successfully logged in, you can edit the book.

(This note thanks to Michele Garoche).

A brief note on Filters

Many a times, you'll want to call a method before or after the performance of a controller action. Filters allow you to do so. Filters conventionally appear at the top of a controller and take either a method name or Proc. Filters are available to all controller classes descending from **`AbstractController`**.

A before filter is called before any of the action code has been executed. They are created using the **`before`** class method which is available to all descendants of **`AbstractController`**. The first parameter of **`before`** can be either a method name or a Proc. Method names may be in the form of strings, but typically are symbols.

We have used a before filter in the above example:

```
before :ensure_authenticated, :exclude => [:show, :index]
```

Merb Console Usage

This note, thanks Michèle Garoche.

Goal:

To provide at a glance how the merb console could be used to test something before putting it in the various locations of a merb application.

Resources:

DataMapper API: http://datamapper.rubyforge.org/
DataMapper documentation: http://datamapper.org/doku.php?
LIFE ON THE EDGE WITH MERB, DATAMAPPER & RSPEC:
http://merb.4ninjas.org/

Prerequisites:

- Shut down the merb server if it is running.

A - What is the merb console for?

- Think of it as the irb equivalent for merb. Apart from executing orders, it may also give you the methods accessible to a given object.
- It executes effectively the orders you launch. If you create new books in a book table, it will effectively create them in the database table.
- *Rollback does not work with DataMapper. Do not try to use the rollback feature with DataMapper, it just does not work.*

B - Launching merb console

You know it already:

`merb -i`
from the top level of your merb application.

Whenever you enter the merb console, the prompt is >>, and the result of a command is prefixed by =>.

C1 - General methods on a given collection

To find all methods available to Order sorted alphabetically, type in the console:
`Order.methods.sort`

It lists a number of methods.

We will explore some of them:

1 - ancestors
`Book.ancestors`

It gives an array of the inheritance class/module hierarchy above Book, included it:

```
[Book,          AuthorsBooksHelper,          DataMapper::Validate,
DataMapper::Timestamp,  #<Module:0x31fcb1c>,  #<Module:0x31fe2f0>,
#<Module:0x31ff984>,       #<Module:0x3200f64>,       Extlib::Hook,
DataMapper::Hook,     DataMapper::Resource,     DataMapper::Types,
DataMapper::Resource::Transaction, DataMapper::Assertions, Object,
Spec::Extensions::Main,     Spec::Expectations::ObjectExpectations,
Base64::Deprecated,                                          Base64,
JSON::Pure::Generator::GeneratorMethods::Object,            Kernel]
```

You'll see here the AuthorsBooksHelper added as a gem previously and included in the Book model. But if you ask for ancestors of Order, you will not find it, as AuthorBooksHelper was not included in the Order model.
`Order.ancestors`

```
[Order, DataMapper::Validate, DataMapper::Timestamp, ...]
```

2 - relationships

`Order.relationships`

It gives a hash of the relationship defined by the has n, and belongs_to in Book and Order models.

```
{:book=>#<DataMapper::Associations::Relationship:0x2499c54
@child_properties=nil, @parent_model="Book", @name=:book,
@parent_properties=nil, @child_model=Order,
@child_key=#<PropertySet:{#<Property:Order:book_id>}>,
@parent_key=#<PropertySet:{#<Property:Book:id>}>,
@options={}, @query={}, @repository_name=:default>}
```

3 - properties
`Order.properties`

It give a hash of the properties defined in the Order model, included the relationship given by belongs_to :book:

```
=>
#<PropertySet:{#<Property:Order:id>,#<Property:Order:custom
er_name>,
 #<Property:Order:price>,#<Property:Order:book_id>}>
```

4 - Others methods
For example, the methods:
`all`, `first`
to retrieve all the rows, or only the first one.

C2 - General methods on a given instance

As there are no orders in the database currently, we will explore some of those methods on the first instance of book.

1 - **`get(1)`** versus **`first`** methods

a - The get(n) method

This method retrieves the item with id n in the table which it is applied on. So, to get the first book registered in the books table, we could use:

```
Book.get(1)
```

And in order to apply methods to this book, we should put the result in a variable, like this:

```
book = Book.get(1)
```

It replies with:

```
~ SELECT "id", "price", "name", "author_id" FROM "books"
WHERE ("id" = 1) ORDER BY "id" LIMIT 1
```

Notice how the get method is transformed into a database request. This works fine provided that there is effectively a book in the database which id is 1. In this case, the result of the command is:

```
=> #<Book id=1 price=2.0 name="The Ultimate Guide to Ruby
Programming" description=#<not loaded#> author_id=1#>
```

Note that the description of the book is not loaded, as we use lazy loading in the Book model on this attribute.

If for some reason, deletion of the first record for example, there is no book with id 1, it will reply with a nil answer:

```
=> nil
```

Then, we will have to resort on another method.

b - The **`first`** method
This method retrieves the first item in the table which it is applied on.

```
>> book=Book.first
 ~ SELECT "id", "price", "name", "author_id" FROM "books"
ORDER BY "id" LIMIT 1
```

```
=> #<Book id=2 price=2.0 name="The Ultimate Guide to Ruby
Programming" description=<not loaded> author_id=1>
```

Here, you'll see that the id of the first item in the table is not 1, but 2, probably because we have destroyed the book with id 1.

You may use whatever method you want, just test that the result it not nil, before applying a method to the result.

Now we are ready to explore some instance methods.

2 – **methods**
This method will retrieve all the methods accessible to a given instance:
book.methods.sort

Notice that the **validates_with_method** methods, such as **positive_price?** method, are listed here.

3 – **attributes**
book.attributes

This method returns a hash of all key-value pairs for the given book, which are the values corresponding to each column in the books table.

```
~ SELECT "description", "id" FROM "books" WHERE ("id" = 1)
ORDER BY "id"
=> {:description=>"The best Ruby guide to get you started
in Ruby", :price=>2.0,
      :author_id=>1, :name=>"The Ultimate Guide to Ruby
Programming", :id=>1}
```

4 - **instance_variables**

book.instance_variables

This method returns an array containing the variables which will be accessible from the views and controller:

```
=> ["@author_id", "@name", "@collection",
"@original_values", "@price", "@new_record", "@errors",
"@id", "@description", "@repository"]
```

5 - Other methods
You have also access to all accessors methods, such as:
`author` and `author=`

As well as methods to create, update or delete a row:
`create`, `save`, `update`, `new`, `destroy`

that have been used previously.

D - Simulating books commands in the console

Now that we have seen some commands, we are going to create some orders from various customers containing possibly multiple copies of several books. Then we will calculate the orders' brut sum for a given customer.

1 - Create orders

To create orders, we need to remind the column names in the Orders table and the available books in the Books table. The all command, which retrieves all the rows in the table, does it for us:

```
>> Order.all
~ SELECT "id", "price", "customer_name", "book_id" FROM
"orders" ORDER BY "id"
=> []
>> Book.all
~ SELECT "id", "price", "name", "author_id" FROM "books"
ORDER BY "id"
=> [#<Book id=1 price=2.0 name="The Ultimate Guide to Ruby
Programming"  description=<not loaded> author_id=1>,
#<Book id=2 price=40.0 name="The Merb Way" description=<not
loaded> author_id=2>,
#<Book id=3 price=8.95 name="Devin's Little Book of Jokes"
description=<not loaded> author_id=3>,
#<Book id=4 price=3.95 name="The Adentures of Tito"
description=<not loaded> author_id=4>,
#<Book id=5 price=2.99 name="Devins' Cookbook"
description=<not loaded> author_id=5>]
>>
```

Now, we will create some orders with the attributes command:

```
>> order = Order.new
=> #<Order id=nil price=nil customer_name=nil book_id=nil>
>> order.attributes = {:id => 1, :price => 0.50,
:customer_name => "Victor Goff",:book_id => 1}
=> {:book_id=>1, :price=>0.5, :customer_name=>"Victor
Goff", :id=>1}
>> order.save
~ INSERT INTO "orders" ("customer_name", "book_id", "id",
"price") VALUES ('Victor Goff', 1, 1, 0.5)
=> true
```

Notice that we do not use the price in the Book catalog, as each customer may have its own particular price.

The other orders to insert in the table will be:

```
>> order = Order.new
>> order.attributes = {:id => 2, :price => 3.95,
:customer_name => "Victor Goff",:book_id => 2}
>> order.save
>> order = Order.new
>> order.attributes = {:id => 3, :price => 40.0,
:customer_name => "George Thompson",:book_id => 2}
>> order.save
>> order = Order.new
>> order.attributes = {:id => 4, :price => 40.0,
:customer_name => "George Thompson",:book_id => 2}
>> order.save
>> order = Order.new
>> order.attributes = {:id => 5, :price => 2.0,
:customer_name => "George Thompson",:book_id => 1}
>> order.save
```

2 - Checking for errors on create

Check each time that the last command order.save returns true. In this case, the order is saved in the table.

If the save command returns false, you may check the errors with the command errors. It returns a hash of all the errors generated while validating the attributes.

Here for example is what happens when the type of price is BigDecimal instead of Float for the price:

```
>> order = Order.new
=> #<Order id=nil price=nil customer_name=nil book_id=nil>
>> order.attributes = {:id => 1, :price => 0.50,
:customer_name => "Victor Goff",:book_id => 1}
=> {:book_id=>1, :price=>0.5, :customer_name=>"Victor
Goff", :id=>1}
>> order.save
=> false
>> order.errors
=> #["Price must be a number"]}>
```

This is the default message on numbers, here it is issued because we supply a Float as price, and a Float is not a BigDecimal.

Warning: There is an issue displaying BigDecimal numbers in views in the context of Merb-DataMappers-SQlite.

3 - **Retrieving orders**
a - **Retrieving all orders**

To retrieve all the orders entered, use the **`all`** command:

```
>> Order.all
~ SELECT "id", "price", "customer_name", "book_id" FROM
"orders" ORDER BY "id"
=> [#<Order id=1 price=0.5 customer_name="Victor Goff"
book_id=1>,
#<Order id=2 price=3.95 customer_name="Victor Goff"
book_id=4>,
#<Order id=3 price=40.0 customer_name="George Thompson"
book_id=2>,
#<Order id=4 price=40.0 customer_name="George Thompson"
book_id=2>,
#<Order id=5 price=2.0 customer_name="George Thompson"
book_id=1>]
```

b - **Retrieving all orders of a given customer**

What we want here is to retrieve all the orders of a given customer. To do it, we could use:

```
>> Order.all(:customer_name => 'Victor Goff')
~ SELECT "id", "price", "customer_name",
"book_id" FROM "orders" WHERE ("customer_name" = 'Victor
Goff') ORDER BY "id"
=> [#<Order id=1 price=0.5 customer_name="Victor Goff"
book_id=1>,
#<Order id=2 price=3.95 customer_name="Victor Goff"
book_id=4>]
```

This command returns all the orders passed by the customer whose name is exactly Victor Goff.

or to save us some typing:

```
>> Order.all(:customer_name.like => '%Vic%')
```

This command returns all the orders passed by customers who have Vic somewhere in their names.

c - **Retrieving all orders of a given customer together with the book's name**.

Now, we want to retrieve the name of the book. To do this, we rely on the book_id, which is mapped to the id in Book table. The command becomes:

```
>> Order.all(:customer_name.like => '%Geor%').map {|u|
[u.customer_name, .price,Book.get(u.book_id).name]}
~ SELECT "id", "price", "customer_name",
"book_id" FROM "orders" WHERE ("customer_name" LIKE
'%Geor%') ORDER BY "id"
~ SELECT "id", "price", "name", "author_id" FROM "books"
WHERE ("id" = 2) ORDER BY "id" LIMIT 1
~ SELECT "id", "price", "name", "author_id" FROM "books"
WHERE ("id" = 2) ORDER BY "id" LIMIT 1
~ SELECT "id", "price", "name", "author_id" FROM "books"
WHERE ("id" = 1) ORDER BY "id" LIMIT 1
=> [["George Thompson", 40.0, "The Merb Way"],
["George Thompson", 40.0, "The Merb Way"],
["George Thompson", 2.0, "The Ultimate Guide to Ruby
Programming"]]
```

d - **Retrieving all orders of a given customer together with the book's name and author's name.**

This time we add the author's first name and last name given by the author_id found in the Book table.

```
Order.all(:customer_name.like => '%Geor%').map {|u|
[u.customer_name, u.price,Book.get(u.book_id).name,

        Author.get(Book.get(order.book_id).author_id).first_
name,

        Author.get(Book.get(order.book_id).author_id).last_n
ame]}

=> [["George Thompson", 40.0, "The Merb Way", "Satish",
"Talim"],
["George Thompson", 40.0, "The Merb Way", "Satish",
"Talim"],
["George Thompson", 2.0, "The Ultimate Guide to Ruby
Programming", "Satish", "Talim"]]
```

4 - **Retrieving number of books bought by customer and sum.**

To get the total price and the number of books bought by a given customer we will use sum and count respectively.

The number of books will be:

```
>> Order.count(:customer_name.like => '%Geor%')
~ SELECT COUNT(*) FROM "orders" WHERE ("customer_name" LIKE
'%Geor%')
=> 3
```

And the total price:

```
>> Order.sum(:price, :customer_name.like => '%Geor%')
~ SELECT SUM("price") FROM "orders" WHERE ("customer_name"
LIKE '%Geor%')
=> 82.0
```

Exercise

Now that you have seen in the merb console how to create orders, retrieve them and calculate the total price, you should be able to implement those features in the application, generating the necessary resources, migrations, validations, and views, as explained in previous sections.

---- The End ----

Resources

A. Thanks to José Carlos Monteiro

- ActiveRecord != DataMapper - The first thing you'll notice is that in DM the properties of the database are in your model, where in AR they are in a migration.
- Rails vs Merb & ActiveRecord/Datamapper - Merb+Datamapper still outshines the competition, but the difference in performance isn't quite as significant, however once the DB is tuned, I bet life gets more interesting...
- Datamapper (or Just Say No to Active Record) - The development philosophy behind Datamapper tries to keep everything as fast and as lightweight as possible. It does this by being as lazy as possible in what it loads to keep memory usage down, and by breaking out as much functionality as possible into plugins, so that you only need to include what you're really using in your application.
- DataMapper - Competition for ActiveRecord? - Where DataMapper may really help Rails developers is in introducing more competition into the Ruby ORM world. By taking some radically different design decisions, particularly with regards to migrations and column specifications DataMapper allows us to see a different, but similarly elegant way of doing things, and hopefully ideas will percolate back into ActiveRecord just as AR has clearly inspired DataMapper.
- Comparing Active Record VS Datamapper...? - I do think AR wins big in the setup department just because it provides pure-ruby drivers for many databases. So you can be running ASAP. We hope to improve on our currently poor platform support in the next few weeks with the release of DataObjects 0.9.0. (We're about half-way there right now.)
- Why active record sucks - As shown, AR is just a wrapper for database access and not appropriate as an ORM. Used carefully, or with very simple applications it really may proof useful.
- ActiveRecord sucks, but Kore Nordmann is wrong - An MVC model component is intended for domain logic, not data access. The description says very clearly that the data access layer is

understood to be underneath or encapsulated by the model. This edict is totally ignored by Rails. The net result of this is people manipulating their model objects inside their controllers as if the models were SQL.

- ActiveRecord does not suck - So I wouldn't say ActiveRecord sucks. I would say that people are expecting ActiveRecord to magically solve their OO design for them, in a false quest to avoid doing the design themselves.
- Active Record vs Objects - So, in the end, I am not against the use of Active Record. I just don't want Active Record to be the organizing principle of the application. It makes a fine transport mechanism between the database and the application; but I don't want the application knowing about Active Records. I want the application oriented around objects that expose behavior and hide data. I generally want the application immune to type changes; and I want to structure the application so that new features can be added by adding new types. (See: The Open Closed Principle)

B. Thanks to RubyInside

- Merb 1.0 Released So Here's 44 Links and Resources To Get You Going

C. Thanks to Takaaki Kato

- http://peepcode.com/products/meet-merb-pdf-draft Comes with a PDF and a 50 minutes video. It's still a draft. We can expect a polished version soon.
- http://lsrc2008.confreaks.com/13-yehuda-katz-merb-the-pocket-rocket-framework.html Katz's Presentation: Merb - The Pocket Rocket Framework
- http://mwrc2008.confreaks.com/04katz.html Katz's Presentation: Faster, Better ORM with DataMapper
- http://rubyhoedown2007.confreaks.com/session02.html Ezra Zygmuntowicz: Exploring Merb
- http://goruco2008.confreaks.com/06_zygmuntowicz.html Ezra Zygmuntowicz: Merb: All you need, nil you don't

D. Thanks to Victor Goff and Tim Kadom

- Atlanta RUG – Introduction to Merb
  http://www.atlrug.org/tkadom/posts/61-ATLRUG-Presentation-Introduction-to-Merb
- Atlanta Merb Repository
  http://github.com/search?q=merbday

E. Thanks to Massimiliano Giroldi

- Yehuda Katz explains Merb
  http://www.infoq.com/interviews/katz-merb

F. Others

- Merb Open Source Book
  http://book.merbist.com/
- Merbcast
  http://www.merbcast.com/index.html

About the Authors

Michèle Garoche is a French woman, 57 years old, living in a charming small city near Paris.

After 20 years experience in Accounting and Finances, she changed to Projects Assisting. She is always programming the necessary tools for the job under various systems and languages. Domains: Engineering, Banks, Mechanics, Chemicals, Aviation, Edition and Printing.

She –

- Occasionally writes articles in computer magazines, translating books or technical notices from English, German, and Italian to French.
- Is involved in translating, debugging, writing, porting and documenting open source software as a hobby since 1988.
- Is currently a mentor at rubylearning.org

Her hobbies include music and theatre.

Satish Talim (http://satishtalim.com/) is a senior software consultant based in Pune, India with over 30+ years of I.T. experience. His experience lies in developing and executing business for high technology and manufacturing industry customers. Personally his strengths lie in Business Development and Business Networking apart from new product and solution ideas. Good experience of organization development. Excellent cross disciplinary background in engineering, computer science and management.

He -

- Has helped start subsidiaries for many US based software companies like **Infonox** based in San Jose, CA
http://www.infonox.com/default.shtml
Maybole Technologies Pvt. Ltd.
http://servient.com/
subsidiary of Servient Inc. (based in Houston, Texas) in Pune, India.

- Has been associated with Java / J2EE since 1995 and involved with Ruby and Ruby on Rails since 2005.
also started and manages two very active Java and Ruby User Groups in Pune, India - **PuneJava**
http://tech.groups.yahoo.com/group/pune-java/
and **PuneRuby**
http://tech.groups.yahoo.com/group/puneruby/

- Is a **Ruby Mentor**
http://rubymentor.rubyforge.org/wiki/wiki.pl?AvailablePureRubyMentors
on rubyforge.org, helping people with Ruby programming.

He lives in Pune, India, with his wife, son and his Labrador Benzy. In his limited spare time he enjoys traveling, photography and playing online chess.

GNU Free Documentation License

Version 1.2, November 2002

```
Copyright (C) 2000,2001,2002  Free Software Foundation,
Inc. 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
USA Everyone is permitted to copy and distribute verbatim
copies of this license document, but changing it is not
allowed.
```

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.