



1 Introduction

This document describes how to provide a simple Rails application. It is for a web application that allows a phone book to be maintained.

The document assumes that a Rails environment has been created. This can be provided by following the commands in either the document *Rails HOW-TO: Installing Rails into Debian* or the document *Rails HOW-TO: Installing Rails into Windows*. These documents are available at <http://www.oucs.ox.ac.uk/rails/howtos>.

2 A phone book

Let us keep things simple: each entry in the phone book is just going to have a name and a number.

3 Creating the Rails application

I'll create a directory for my Rails applications.

```
mkdir /var/apps
cd /var/apps
```

You use the `rails` command to create a new Rails application. By default, it creates an application that uses SQLite3. You can use the `-d` option to use some other database server.

The following command creates a directory called `contacts` and creates a lot of subdirectories/files in that directory.

```
rails -d mysql contacts
cd /var/apps/contacts
ls
```

One file is called `README`: it includes an explanation of the purpose of the subdirectories.

4 Configuring Rails's access to MySQL

One subdirectory is called `config`. That directory contains a file called `database.yml`. This is the configuration for this application's access to the chosen database server.

By default, it assumes you want three databases: one for development, one for testing and one for production. It also assumes your web application is going to contact the database server on the standard port (3306 for MySQL) using a username of `root` with an empty password.

Because I want some different values, I'm going to provide a different file.

```
cd /var/apps/contacts/config
cat database.yml
cat >database.yml <<%
development:
  adapter: mysql
  port: 8116
  encoding: utf8
  database: contacts_development
  username: ruby
  password: PW4ruby
# Warning: The database defined as 'test' will be erased and
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
```

```
adapter: mysql
port: 8116
encoding: utf8
database: contacts_test
username: ruby
password: PW4ruby
production:
  adapter: mysql
  port: 8116
  encoding: utf8
  database: contacts_production
  username: ruby
  password: PW4ruby
%
```

5 Configuring permissions in MySQL

Having chosen to use a particular username and password, we also need to inform MySQL that these three databases are to be accessed in this way.

```
mysql -P 8116 -u root -p mysql <<%
grant all privileges on contacts_development.* \
  to ruby@localhost identified by 'PW4ruby';
grant all privileges on contacts_production.* \
  to ruby@localhost identified by 'PW4ruby';
grant all privileges on contacts_test.* \
  to ruby@localhost identified by 'PW4ruby';
flush privileges;
\q
%
PW4root
```

6 Getting rails to create the databases

The Rails software has a command called `rake` that behaves like Unix's `make` command. The following `rake` command gets MySQL to create the three databases.

```
cd /var/apps/contacts
rake db:create:all
```

Note: if I hadn't wanted to use a different port, username and password, we could have jumped from creating the Rails application to this stage.

7 Creating the scaffold

Ruby comes with a number of scripts that do useful things. These are in a subdirectory called `script`. One of these (called `generate`) can be used to create a *scaffold*. A scaffold is an initial stab at all the files you need for a web application that manipulates data stored in some table of the database.

We need to give the scaffold-building script the name of the class and the names and types of the attributes of the class.

```
cd /var/apps/contacts
ruby script/generate scaffold Phone \
  name:string \
  number:string
```

To keep things simple, I have chosen to use the type `string` for both attributes. Other possibilities include `binary`, `boolean`, `date`, `datetime`, `decimal`, `float`, `integer`, `text`, `time` and `timestamp`.

8 A migration file

Currently, we have an empty database. With Rails, you can change a database by performing a migration. One of the files the above command creates is one with a name like `db/migrate/20080615150932_create_phones.rb` where the actual name depends on the current date and time.

Note: with versions of Rails prior to Rails 2.1, this file would have a name like `db/migrate/001_create_phones.rb`

```
cd /var/apps/contacts/db/migrate
cat *_create_phones.rb
```

Here is its contents.

```
class CreatePhones < ActiveRecord::Migration
  def self.up
    create_table :phones do |t|
      t.string :name
      t.string :number
      t.timestamps
    end
  end
  def self.down
    drop_table :phones
  end
end
```

It can be used to add/remove the table to/from the database.

9 Migrating the database

Here is the command to do a migration.

```
cd /var/apps/contacts
rake db:migrate
```

This command looks in the `db/migrate` directory to see if there are any files which have not yet been executed. It will execute each of these in turn. In our example, it just finds `20080615150932_create_phones.rb` and it executes this file's `up` method.

We can check what has happened using the following.

```
mysql -P 8116 -u root -p contacts_development <<%
show create table phones;
select * from phones;
\q
%
PW4root
# I got no output from the select because the table exists but is empty
```

If you were now to do a command like:

```
rake db:migrate:down VERSION=20080615150932
```

it would execute the `down` method of this file.

10 Starting the WEBrick server

Ruby comes with some web server software called WEBrick. It knows how to run Ruby applications. In the subdirectory called `script`, you will find a file called `server`. This can be used to start WEBrick.

By default, WEBrick listens on port 3000 but I want to use port 8119. I will include an ampersand on the command line in order to start the web server as a background process.

```
cd /var/apps/contacts
ruby script/server -p 8119 &
# I got:
# => Booting WEBrick...
# => Rails application started on http://0.0.0.0:8119
# => Ctrl-C to shutdown server; call with --help for options
# [2008-05-31 12:16:11] INFO WEBrick 1.3.1
# [2008-05-31 12:16:11] INFO ruby 1.8.5 (2006-08-25) [i486-1]
# [2008-05-31 12:16:11] INFO WEBrick::HTTPServer#start: pid=28586 port=8119
```

Note: this output says that WEBrick is running in process 28586.

We can use a `ps` command to check the server is running.

```
ps -ef | grep ruby
```

11 Testing the Rails application

We can now test the Rails application.

```
http://www.abcd.ox.ac.uk:8119/phones
```

We can add a new phone number using the URL:

```
http://www.abcd.ox.ac.uk:8119/phones/new
```

After you have added a few phone numbers, you will see that each phone number has its own `id`. We can show the details of a particular phone number using that `id`:

```
http://www.abcd.ox.ac.uk:8119/phones/1
```

And we can edit an entry using a URL like:

```
http://www.abcd.ox.ac.uk:8119/phones/1/edit
```

12 Killing the WEBrick process

Earlier, we used a `ps` command to check whether WEBrick is running.

```
ps -ef | grep ruby
```

We got output like:

```
chris 28586 11254 0 09:33 pts/6 00:00:02 ruby script/server -p 8119
chris 4366 31190 0 10:39 pts/11 00:00:00 grep ruby
```

This means that WEBrick is running in process 28586.

We can stop the WEBrick server from running by killing this process. This can be done by using the following command where PPP is replaced by the appropriate process number (e.g., 28586).

```
kill -KILL PPP
```

13 Model View Controller

13.1 Rails uses Model View Controller

Rails exploits a programming technique called *Model View Controller* (or *MVC*).

Here the *Model* is concerned with the data you want to represent. So it could be a person, or a person's name, or a CD, or a bank account, or a grid reference,

The *Controller* has the code that causes the data of the Model to change.

And the *View* is the code that shows the results of an action. Typically, these files have all the code that generates HTML (or XML or ...).

13.2 Establishing a model

At the top level of a Rails app, there is a directory called `app`. In this directory, there are subdirectories called `models`, `controllers` and `views`.

By default, a pattern called *ActiveRecord* is used to establish the Model: a class in the program is mapped to the schema of a database table and each instance of the class (i.e., each object) is stored in a different row of the table. So, with simple examples, there is little code in the `models` directory as the Model is described by the table of the database.

For this example, the database is called `contacts_development`, the table is called `phones` and the associated Ruby class is called `Phone`.

13.3 Establishing a controller

The crucial file of the `controllers` directory has a name like `phones_controller.rb`. This has one method for each action. Here is an example.

```
class PhonesController < ApplicationController
  def index
    @phones = Phone.find(:all)
    respond_to do |format|
      format.html # index.html.erb
      format.xml { render :xml => @phones }
    end
  end
  ...
end
```

The final part of the code of each method indicates what view to use (to show the result of the action).

The request (for the action) will indicate the format of the response. It will often be HTML or XML.

13.4 Establishing the views

The `views` directory has a subdirectory (with a name like `phones`) which has one file for each view.

For example, there is a file called `index.html.erb` that contains the code of the web page that just lists the items of the table.

```
<h1>Listing phones</h1>
<table>
  <tr>
    <th>Name</th>
    <th>Number</th>
  </tr>
  <% for phone in @phones %>
  <tr>
    <td><%=h phone.name %></td>
    <td><%=h phone.number %></td>
    <td><%= link_to 'Show', phone %></td>
    <td><%= link_to 'Edit', edit_phone_path(phone) %></td>
    <td><%= link_to 'Destroy', phone,
      :confirm => 'Are you sure?',
      :method => :delete %></td>
  </tr>
```

```
<% end %>
</table>
<br />
<%= link_to 'New phone', new_phone_path %>
```

Here are some notes about the above code:

- The HTML given here is embedded in the layout file `.../app/views/layout/phones.html.erb`.
- The `<% ... %>` identifies code written in Ruby.
- The `<%= ... %>` causes the output of some Ruby code to be written into the HTML being generated.
- Instance variables established in the controller (e.g., `@phones`) are automatically established as instance variables in each view (e.g., `@phones`).
- The `h phone.name` is a call of the `h` method with one argument (`phone.name`). The `h` method does HTML escaping. So if `phone.name` were to contain a `<`, it will be output as `<`.
- The `link_to` method is a *helper method* that creates a link element in HTML (i.e., an `a` element where the first argument (e.g., `'Show'`) is the label of the link and the other arguments (e.g., `phone`) are used to construct the URL.
- `edit_phone_path` and `new_phone_path` are two methods that are automatically created by Instant Rails.

14 Conclusions

We have built a web application that maintains some data, a phone book where each entry just has a name and a number.

However, note that most of the hard work was done by Rails:

- it looked after the SQL needed to modify the database;
- it wrote the code of the model, the controller and the views.