



## 1 Introduction

A quick introduction to providing a simple Rails application is given in the document *Rails HOW-TO: A simple Rails apps: phones*. That document is available at <http://www.oucs.ox.ac.uk/rails/howtos>.

However, that document leaves out many details. Details about some other aspects I have found useful are covered in this document.

This document looks at how to:

- validate the data before it is stored;
- *clean* the data before it gets validated;
- customise the HTML that gets used on the web pages;
- provide other actions besides those provided by the scaffold.

This document assumes the reader has read the document *Rails HOW-TO: A simple Rails apps: phones*.

## 2 Validating the data

It is important that data is validated before it is stored. This is to ensure that any data calculated by our program or provided by the user has an appropriate value. In Rails, the class `ActiveRecord::Base` provides a lot of methods that help you do this.

For example, we can indicate that we only want a numerical value for the phone number by using the method call:

```
validates_numericality_of :number
```

This is a call of a method called `validates_numericality_of` passing to it the name of the attribute to be checked (`:number`). If the value has to be an integer, you can use an `only_integer` option to indicate this.

```
validates_numericality_of :number, :only_integer => true
```

This method call should be included in the class for the Model:

```
class Phone < ActiveRecord::Base
  validates_numericality_of :number, :only_integer => true
end
```

There are a lot of validation methods (besides `validates_numericality_of`).

Another one is `validates_format_of`. This enables you to indicate that a particular attribute must match a regular expression. For example, we could add the following to the class declaration for `Phone`: it says that the `name` attribute must consist of two alphabetic strings separated by a space:

```
validates_format_of :name, :with => /[a-z]+ [a-z]+/i
```

A full list of validation methods is given at

<http://rails.rubyonrails.com/classes/ActiveRecord/Validations/ClassMethods.html>

### 3 Filtering the input values

It may be that you want to be relaxed about how the user types a value but that you want the value to be *cleaned* before it gets stored.

For example, suppose we want to allow the user to type spaces when supplying a phone number but that we want these spaces to be removed before the phone number gets validated. We can do this by using a method called `before_validation` that requests Ruby to execute some code before it does validation. We can call `before_validation` with a parameter that gives the name of the method that contains the code that does the cleaning.

Here is an example:

```
class Phone < ActiveRecord::Base
  before_validation :remove_spaces_from_number
  validates_format_of :name, :with => /[a-z]+ [a-z]+/i
  validates_numericality_of :number, :only_integer => true
  private
  def remove_spaces_from_number
    self.number = self.number.gsub(/ /, "")
  end
end
```

## 4 Customising the web pages

### 4.1 Changing the layout

Files like `index.html.erb`, `show.html.erb`, ... contain templates for a particular view. They get used by the layout template. This is stored in a file with a name like:

```
.../app/views/layouts/phones.html.erb
```

This file contains the skeleton for a web page. So it starts with:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
```

and ends with:

```
</body>
</html>
```

So, when a view is presented, it is this web page that is used. However, in the middle of this HTML there is a line of Embedded Ruby:

```
<%= yield :layout %>
```

This is magic that says use the template for the view.

So, if the view is the result of the `index` action, the file `index.html.erb` will be used at this point.

The file `.../app/views/layouts/phones.html.erb` is the file you need to alter if you want to change the look of this web site. It is here that you can provide branding, menus and navigation.

### 4.2 Using a partial

If you have a portion of HTML (or Embedded Ruby) that gets used in a number of pages, then (to ensure you Don't Repeat Yourself), you will want to provide this code once. One way of doing this is to use a *partial*.

A partial is a file containing a fragment of Embedded Ruby. The fragment is provided in a file whose name begins with an underscore, such as `_output_size.html.erb`. It could contain some code such as:

```
<div class="output_size">
  <p>number of phones is <%=h size %></p>
</div>
```

In each template file where you want this to appear, include:

```
<%= render :partial => 'output_size',
         :locals => { :size => @number_of_phones } %>
```

This code assumes the view has access to an instance variable called `@number_of_phones`. You could get the controller to create an appropriate instance variable by ensuring that `phones_controller.rb` starts with something like:

```
class PhonesController < ApplicationController
  before_filter :find_number_of_phones
```

and finishes with something like:

```
  private
  def find_number_of_phones
    @number_of_phones = Phone.find(:all).size()
  end
end
```

We now look at another example of the use of partials. The forms given in the files `new.html.erb` and `edit.html.erb` contain similar code. So the code in `new.html.erb` could be replaced by:

```
<%= render :partial => 'fields',
         :locals => { :phone => @phone, :label => 'Create' } %>
```

and the code in `edit.html.erb` could be replaced by:

```
<%= render :partial => 'fields',
         :locals => { :phone => @phone, :label => 'Update' } %>
```

Both of these refer to the partial in the file `_fields_html.erb`. This file could contain something like:

```
<% form_for(phone) do |f| %>
  <%= render
    :partial => 'field',
    :locals =>
      { :key => 'Name', :value => f.text_field(:name) } %>
  <%= render
    :partial => 'field',
    :locals =>
      { :key => 'Number', :value => f.text_field(:number) } %>
  <p>
    <%= f.submit label %>
  </p>
<% end %>
```

where the file `_field.html.erb` contains:

```
<p>
  <b><%= key %>:</b>
  <%= value %>
</p>
```

Similarly, the file `show.html.erb` could be altered to use the `field` partial twice.

## 5 Providing other actions besides those provided by the scaffold

### 5.1 Introduction

Suppose we want to add a new action to the phone book that allows us to search for a name that matches a string. So a request like:

```
http://localhost:3000/phones/search/*row*
```

will find all the entries of the phone book that have the characters `row` somewhere in the name.

There are three things that need to be done to achieve this:

- alter the routing so that the URL will be dispatched to the controller;
- alter the controller to provide a new method that finds the entries that match the search;
- provide a view to display the results.

### 5.2 Alter the routing so the URL is dispatched

We are adding a new action to the Rails application. It is one called `search`. So we have to alter the routing code so that a URL with an action of `search` is passed to the `search` method of the controller.

The routing code is given in the file `.../app/config/routes.rb`. The crucial line of this file is the one that says:

```
map.resources :phones
```

This line magically gives us seven routes in to our resource: it handles GET, POST, DELETE and PUT requests using URLs like:

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

and

```
http://localhost:3000/phones/2.xml
```

Note: if you want to see the specific details about the seven routes, look at the second table in the document *Rails HOW-TO: Using REST*. That document is available at <http://www.oucs.ox.ac.uk/rails/howtos>.

The `map.resources` line can be altered to:

```
map.resources :phones, :collection => hash1, :member => hash2
```

You use `hash1` if you want to add an action that manipulates the whole table and you use `hash2` if you want to add an action that manipulates a specific item, i.e., a particular row of the table.

As we want an action that searches the whole table, we will add a value to the first hash. We have to supply the name of the action and what kind of HTTP request is used to request it.

```
map.resources :phones, :collection => { :search => :get }
```

Here are some commands that achieve this edit.

```
cd /var/apps/contacts
head -12 config/routes.rb
ed config/routes.rb <<'%'
g/map.resources :phones/s/$/, :collection => { :search => :get }/p
w
q
%
head -12 config/routes.rb
```

### 5.3 Alter the controller to provide a new method

We need to provide some code in the controller for a method called `search`. This code needs to be added to the file `.../app/controllers/phones_controller.rb`.

```
cd /var/apps/contacts
ed app/controllers/phones_controller.rb <<'%'
$i
def search
  pattern = params[:id].gsub(/\*/, "%")
  @phones = Phone.find(:all,
    :conditions => ["name like ?", pattern],
    :order => "name ASC"
  )
  respond_to do |format|
    format.html # search.html.erb
    format.xml { render :xml => @phones }
  end
end
end
.
w
q
%
```

### 5.4 Provide a view to display the results

We need to provide a view that shows the results of the search. This needs to be provided in the file `.../app/views/phones/search.html.erb`. The easiest way of creating this file is to copy the code for showing the index and modify its heading.

```
cd /var/apps/contacts
ed app/views/phones/index.html.erb <<'%'
g/<h1>/s;.*;<h1>Result of searching for phones</h1>;p
w app/views/phones/search.html.erb
q
%
```

### 5.5 Testing the search action

If we now use a browser with the URL

```
http://localhost:3000/phones/search/*row*
```

it should display all the entries of the phone book that have the characters `row` somewhere in the name.

And if we use the URL

```
http://localhost:3000/phones/search/*own
```

it should display all the entries of the phone book that have a name ending with the characters `own`.

And the URL

```
http://localhost:3000/phones/search/br*
```

should display all the entries of the phone book that have a name starting with the characters `br`.