Four Days on Rails

compiled by John McCree

Some Rights Reserved
# Table of Contents

**Introduction** .................................................................................................................. 1

**Day 1 on Rails** .................................................................................................................. 3

- The ‘To Do List’ application .......................................................................................... 3
- Running the Rails script .................................................................................................. 3
- Adding the Application to the Web Server ..................................................................... 3
  - Defining the Application in the hosts file .................................................................... 3
  - Defining the Application in the Apache Configuration file ...................................... 3
  - Switching to fastcgi ...................................................................................................... 3
  - Checking that Rails is working ...................................................................................... 4
  - Versions of Rails .......................................................................................................... 4
- Setting up the Database .................................................................................................... 4
  - Creating the Categories Table ...................................................................................... 4
    - MySQL definition .......................................................................................................... 4
    - Data Model .................................................................................................................. 5
  - Scaffold .......................................................................................................................... 9
  - Enhancing the Model ...................................................................................................... 6
  - Creating Data Validation Rules .................................................................................... 6

**Day 2 on Rails** .................................................................................................................. 9

- The Generated Scaffold Code .......................................................................................... 9
  - The Controller .............................................................................................................. 9
  - The View ...................................................................................................................... 10
    - Layout ......................................................................................................................... 11
    - Template ..................................................................................................................... 11
    - Partial .......................................................................................................................... 12
  - The Rendered View for the “New” action ..................................................................... 13
  - Analysing the View for the ‘List’ action ...................................................................... 13
- Tailoring the Generated Scaffold Code ............................................................................ 15
  - The Controller .............................................................................................................. 15
  - The View ...................................................................................................................... 15
    - Displaying Flash Messages ....................................................................................... 15
    - Sharing Variables between the Template and Layout .............................................. 16
  - Tidying up the Edit and New Screens .......................................................................... 17

**Day 3 on Rails** .................................................................................................................. 19

- The ‘Items’ Table .............................................................................................................. 19
  - MySQL table definition ............................................................................................. 19
  - The Model ..................................................................................................................... 19
    - Validating Links between Tables .............................................................................. 20
    - Validating User Input ................................................................................................. 20
- The ‘Notes’ table ............................................................................................................... 20
  - MySQL table definition ............................................................................................. 20
  - The Model ..................................................................................................................... 20
    - Using a Model to maintain Referential Integrity ..................................................... 21
- More Scaffolding .............................................................................................................. 21
- More on Views ................................................................................................................ 21
  - Creating a Layout for the Application ......................................................................... 21
  - The ‘To Do List’ screen ............................................................................................... 21
    - Purging completed ‘To Dos’ by clicking on an icon ............................................... 23
    - Changing the Sort Order by clicking on the Column Headings ......................... 24
    - Adding a Helper ........................................................................................................ 24
    - Using Javascript Navigation Buttons ..................................................................... 25
    - Formatting a Table with a Partial ......................................................................... 25
    - Formatting based on Data Values ......................................................................... 26
    - Handling Missing Values in a Lookup .................................................................. 26
  - The ‘New To Do’ Screen .............................................................................................. 26
Creating a Drop-down List for a Date Field........................................................................................................ 27
Trapping Exceptions in Ruby........................................................................................................................................ 27
Creating a Drop-down List from a Lookup Table................................................................................................ 28
Creating a Drop-down List from a List of Constants............................................................................................... 28
Creating a Checkbox.................................................................................................................................................... 28
Finishing Touches........................................................................................................................................................ 28
Tailoring the Stylesheet.................................................................................................................................................. 28
The ‘Edit To Do’ Screen.............................................................................................................................................. 29

Day 4 on Rails............................................................................................................................................................... 31
The ‘Notes’ screens....................................................................................................................................................... 31
  Linking ‘Notes’ to the ‘Edit To Do’.............................................................................................................................. 31
  The ‘Edit Notes’ Screen................................................................................................................................................ 32
  The ‘New Note’ Screen................................................................................................................................................. 32
  Saving and retrieving Data using Session Variables................................................................................................ 33
Changing the ‘Categories’ Screens.................................................................................................................................. 33
Navigation through the system....................................................................................................................................... 34
  Setting the Home Page for the Application.................................................................................................................. 35
  Downloading a Copy of this Application........................................................................................................................ 35
  and finally...................................................................................................................................................................... 35

Appendix – afterthoughts............................................................................................................................................. 37
Multiple Updates.............................................................................................................................................................. 37
  View.................................................................................................................................................................................. 37
  Controller....................................................................................................................................................................... 38
  User Interface considerations........................................................................................................................................... 39
  Still to be done............................................................................................................................................................. 39
Introduction

There have been many extravagant claims made about Rails. For example, an article in OnLAMP.com\(^1\) claimed that “you could develop a web application at least ten times faster with Rails than you could with a typical Java framework...” The article then went on to show how to install Rails and Ruby on a PC and build a working ‘scaffold’ application with virtually no coding.

While this is impressive, ‘real’ web developers know that this is smoke and mirrors. ‘Real’ applications aren’t as simple as that. What’s actually going on beneath the surface? How hard is it to go on and build ‘real’ applications?

This is where life gets a little tricky. Rails is well documented on-line – in fact, possibly too well documented for beginners, with over 30,000 words of on-line documentation in the format of a reference manual. What’s missing is a roadmap (railmap?) pointing to the key pages that you need to know to get up and running in Rails development.

This document sets out to fill that gap. It assumes you’ve got Ruby and Rails up on a PC (if you haven’t got this far, go back and follow Curt’s article). This takes you to the end of ‘Day 1 on Rails’.

‘Day 2 on Rails’ starts getting behind the smoke and mirrors. It takes you through the ‘scaffold’ code. New features are highlighted in bold, explained in the text, and followed by a reference to either Rails or Ruby documentation where you can learn more.

‘Day 3 on Rails’ takes the scaffold and starts to build something recognisable as a ‘real’ application. All the time, you are building up your tool box of Rails goodies. Most important of all, you should also be feeling comfortable with the on-line documentation so you can continue your explorations by yourself.

‘Day 4 on Rails’ adds in another table and deals with some of the complexities of maintaining relational integrity. At the end, you’ll have a working application, enough tools to get you started, and the knowledge of where to look for more help.

Ten times faster? after four days on Rails, judge for yourself!

Documentation: this document contains highlighted references, either to:

- **Documentation** – the Rails documentation at \[http://api.rubyonrails.com\](http://api.rubyonrails.com) (this documentation is also installed on your PC as part of your gems installation in a location like `C:\Program Files\ruby\lib\ruby\gems\n.n\doc\actionpack-n.n.n\zdoc\index.html`)

Acknowledgements: many thanks to the helpful people on the the irc channel\(^2\) and the mailing list\(^3\). The online archives record their invaluable assistance as I clawed my way up the Rails and Ruby leaning curves.


Copyright: this work is copyright ©2005 John McCreesh \[jpmcc@users.sourceforge.net\](mailto:jpmcc@users.sourceforge.net) and is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit \[http://creativecommons.org/licenses/by-nc-sa/2.0/\](http://creativecommons.org/licenses/by-nc-sa/2.0/) or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

---


\(^2\) \[irc://irc.freenode.org/rubyonrails\](irc://irc.freenode.org/rubyonrails)

Day 1 on Rails

The ‘To Do List’ application

This document follows the building of a simple ‘To Do List’ application – the sort of thing you have on your PDA, with a list of items, grouped into categories, with optional notes (for a sneak preview of what it will look like, see Illustration 5: The ‘To Do List’ Screen on page 23).

Running the Rails script

This example is on my MS-Windows PC. My web stuff is at c:\www\webroot, which I label as drive w: to cut down on typing:

C:\> subst w: c:\www\webroot
C:\> w:
W:\> rails ToDo
W:\> cd ToDo
W:\ ToDo>

Running rails ToDo creates a new directory ToDo\ and populates it with a series of files and subdirectories, the most important of which are as follows:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>contains the core of the application, split between model, view, controller, and ‘helper’ subdirectories</td>
</tr>
<tr>
<td>config</td>
<td>contains the database.yml file which provides details of the database to used with the application</td>
</tr>
<tr>
<td>log</td>
<td>application specific logs. Note: development.log keeps a trace of every action Rails performs – very useful for error tracking, but does need regular purging!</td>
</tr>
<tr>
<td>public</td>
<td>the directory available for Apache, which includes images, javascripts, and stylesheets subdirectories</td>
</tr>
</tbody>
</table>

Adding the Application to the Web Server

As I’m running everything (Apache2, MySQL, etc) on a single development PC, the next two steps give a friendly name for the application in my browser.

Defining the Application in the hosts file

C:\winnt\system32\drivers\etc\hosts (excerpt)

127.0.0.1 todo

Defining the Application in the Apache Configuration file

Apache2\conf\httpd.conf

<VirtualHost *>
    ServerName todo
    DocumentRoot /www/webroot/ToDo/public
    <Directory /www/webroot/ToDo/public/>
    Options ExecCGI FollowSymLinks
    AllowOverride all
    Allow from all
    Order allow,deny
    </Directory>
</VirtualHost>

Switching to fastcgi

Unless you are patient (or have a powerful PC) you should enable fastcgi for this application
Checking that Rails is working
The site should now be visible in your browser as http://todo/ (you should see the Congratulations, you've put Ruby on Rails! page in your browser).

Versions of Rails
By the time you read this document, Rails will probably have moved on several versions. If you intend to work through this document, check the versions installed on your PC:

```
W:\ToDo>gem list --local
```

If they are different from the versions listed below, then I would strongly advise you to download the versions used in 'Four Days', e.g:

```
W:\ToDo>gem install rails --version 0.12.1
```

This won’t break anything; Ruby’s gems library is designed to handle multiple versions. You can then force Rails to use the ‘Four Days’ versions with the ‘To Do List’ application by specifying:

```
config\environment.rb (excerpt)
# Require Rails libraries.
require 'rubyrails'
require_gem 'activesupport', '= 1.0.4'
require_gem 'activerecord', '= 1.10.1'
require_gem 'actionpack', '= 1.8.1'
require_gem 'actionmailer', '= 0.9.1'
require_gem 'actionwebservices', '= 0.7.1'
require_gem 'rails', '= 0.12.1'
```

The reason using the same versions is quite simple. ‘Four Days’ uses a lot of code generated automatically by Rails. As Rails develops, so does this code – unfortunately, this document doesn’t (until I get round to producing a new version!). So, make life easy for yourself, and keep to the same versions as used in ‘Four Days’. Once you’ve finished working through ‘Four Days’, by all means go onto the latest and greatest Rails versions and see what improvements the Rails developers have come up with.

Setting up the Database
I’ve set up a new database called ‘todos’ in MySQL. Connection to the database is specified in the config\database.yml file

```
config\database.yml (excerpt)
development:
  adapter: mysql
  database: todos
  host: localhost
  username: foo
  password: bar
```

Creating the Categories Table
The categories table is used in the examples that follow. It’s simply a list of categories that will be used to group items in our To Do list.

```
MySQL definition
Categories table
CREATE TABLE `categories` (
  `id` smallint(5) unsigned NOT NULL auto_increment,
```
Some hints and gotchas for table and field naming:

- underscores in field names will be changed to spaces by Rails for ‘human friendly’ names
- beware mixed case in field names – some parts of the Rails code have case sensitivities
- every table should have a primary key called `id` - in MySQL it’s easiest to have this as numeric auto_increment
- links to other tables should follow the same ‘_id’ naming convention
- Rails will automatically maintain fields called `created_at`/`created_on` or `updated_at`/`updated_on`, so it’s a good idea to add them in

**Documentation: ActiveRecord::Timestamp**

- Useful tip: if you are building a multi-user system (not relevant here), Rails will also do optimistic locking if you add a field called `lock_version` (integer default 0). All you need to remember is to include `lock_version` as a hidden field on your update forms.

**Documentation: ActiveRecord::Locking**

**Data Model**

Generate an empty file:

```ruby
W:\ToDo>ruby script/generate model category
exists  app/models/
exists  test/unit/
exists  test/fixtures/
create  app/models/category.rb
create  test/unit/category_test.rb
create  test/fixtures/categories.yml
W:\ToDo>
```

which creates an empty `category.rb`, and two test files `category_controller_test.rb` and `categories.yml`. We’ll make some entries in the data model in a minute – leave it empty just now.

**Scaffold**

The controller is at the heart of a Rails application.

**Running the generate controller script**

```ruby
W:\ToDo>ruby script/generate controller category
exists  app/controllers/
exists  app/helpers/
create  app/views/category
exists  test/functional/
create  app/controllers/category_controller.rb
create  test/functional/category_controller_test.rb
create  app/helpers/category_helper.rb
W:\ToDo>
```

which creates two files and two empty directories:

```
app\controllers\category_controller.rb
app\helpers\category_helper.rb
app\views\categories
app\views\layouts
```

If you haven’t already seen the model / scaffold trick in operation in a beginner’s tutorial like *Rolling with Ruby on Rails*, try it now and amazed yourself how a whole web app can be written in one line of code:
Point your browser at http://todo/category and marvel at how clever it is :-)
To try this out, now try to insert a duplicate record again. This time, Rails handles the error rather than crashing - see below. The style is a bit in your face – it's not the most subtle of user interfaces. However, what do you expect for free?

**New category**

1 error prohibited this category from being saved

There were problems with the following fields:

- Category already exists

Category

Business

Created on

Updated on

Create

Back

Illustration 2: Capturing data errors
Day 2 on Rails

To progress beyond this point, we need to see what’s happening behind the scenes. During day 2, we will work systematically through the scaffold code generated by Rails, deciphering what it all means. With the scaffold action, Rails generates all the code it needs dynamically. By running scaffold as a script, we can get all the code written to disk where we can investigate it and then start tailoring it to our requirements.

**Running the generate scaffold script**

W:\ToDo>ruby script/generate scaffold category

dependency model
dependency exists app/models/
dependency exists test/unit/
dependency exists test/fixtures/
dependency skip app/models/category.rb
dependency skip test/unit/category_test.rb
dependency skip test/fixtures/categories.yml
dependency exists app/controllers/
dependency exists app/helpers/
dependency create app/views/categories
dependency exists test/functional/
dependency create app/controllers/categories_controller.rb
dependency create test/functional/categories_controller_test.rb
dependency create app/helpers/categories_helper.rb
dependency create app/views/layouts/categories.rhtml
dependency create public/stylesheets/scaffold.css
dependency create app/views/categories/list.rhtml
dependency create app/views/categories/show.rhtml
dependency create app/views/categories/new.rhtml
dependency create app/views/categories/edit.rhtml
dependency create app/views/categories/_form.rhtml

W:\ToDo>

This script generates a range of files needed to create a complete application, including a controller, views, layouts, and even a style sheet.

Note the slightly bizarre naming convention – we’ve moved from the singular to the plural, so to use the new code you need to point your browser at http://todo/categories. In fact, to avoid confusion, it’s best to delete app\controllers\category_controller.rb etc in case you run it accidentally.

**The Generated Scaffold Code**

**The Controller**

Let’s look at the code behind the controller. The controller is where the programming logic for the application lies. It interacts with the user using views, and with the database through models. You should be able to read the controller and see how the application hangs together.

The controller produced by the generate scaffold script is listed below:

```ruby
\app\controllers\categories_controller.rb

class CategoriesController < ApplicationController
  def index
    list
    render_action 'list'
  end

  def list
    @category_pages, @categories = paginate :category, :per_page => 10
  end

  def show
    @category = Category.find(@params[:id])
  end

  def new
```
When the user of a Rails application selects an action – e.g. 'Show' – the controller will execute any code in the appropriate section – 'def show' – and then by default will render a template of the same name - 'show.rhtml'. This default behaviour can be overwritten:

- **render_template** allows you to render a different template – e.g. the index action will run the code for 'list' - 'def list', and will then render list.rhtml rather than index.rhtml (which doesn’t exist)
- **redirect_to** goes one stage further, and uses an external ‘302 moved’ HTTP response to loop back into the controller – e.g. the destroy action doesn’t need to render a template. After performing its main purpose (destroying a category), it simply takes the user to the list action.

The controller uses ActiveRecord methods such as find, find_all, new, save, update_attributes, and destroy to move data to and from the database tables. Note that you do not have to write any SQL statements, but if you want to see what SQL Rails is using, it’s all written to the development.log file.

Notice how one logical activity from the user’s perspective may require two passes through the controller: for example, updating a record in the table. When the user selects ‘Edit’, the controller extracts the record they want to edit from the model, and then renders the edit view. When the user has finished editing, the edit view invokes the update action, which updates the model and then invokes the show action.

**The View**

Views are where the user interface are defined. Rails can render the final HTML page presented to the user from three components:
<table>
<thead>
<tr>
<th>Layout</th>
<th>Template</th>
<th>Partial</th>
</tr>
</thead>
<tbody>
<tr>
<td>in app\views\layouts\ default: application.rhtml or &lt;controller&gt;.rhtml</td>
<td>in app\views&lt;controller&gt;\ default: &lt;action&gt;.rhtml</td>
<td>in app\views&lt;controller&gt;\ default _&lt;partial&gt;.rhtml</td>
</tr>
</tbody>
</table>

- A Layout provides common code used by all actions, typically the start and end of the HTML sent to the browser.
- A Template provides code specific to an action, e.g. ‘List’ code, ‘Edit’ code, etc.
- A Partial provides common code - ‘subroutines’ - which can be used in used in multiple actions – e.g. code used to lay out tables for a form.

**Layout**

Rails Naming conventions: if there is a template in app\views\layouts\ with the same name as the current controller then it will automatically be set as that controller’s layout unless explicitly told otherwise.

A layout with the name application.rhtml or application.rxml will be set as the default controller if there is no layout with the same name as the current controller, and there is no layout explicitly assigned.

The layout generated by the scaffold script looks like this:

```html
<%= @content_for_layout %>
```
The new .rhtml created by the scaffold script is given below:

```html
app\views\categories\new.rhtml
<h1>New category</h1>
<%=
  start_form_tag :action => 'create'
%>
<%=
  render_partial "form" %>
<%=
  submit_tag "Create" %>
<%=
  end_form_tag %>
<%=
  link_to 'Back', :action => 'list'
%>
```

- `start_form_tag` is a Rails helper to start an HTML form – here it generates `<form action="/categories/create" method="post">`
- `submit_tag` by itself would generate `<input name="submit" type="submit" value="Save changes" />`, but the “Create” parameter overwrites the default “Save changes” with “Create”
- `end_form_tag` just outputs `<form>`, which is not the most useful Rails helper ever written :-) but it provides a satisfying end to the block of code

**Documentation:** `ActionView::Helpers::FormTagHelper`

- `render_partial` will invoke a Partial _form.rhtml_ – see the next section.

**Documentation:** `ActionView::Partials`

- `link_to` simply creates a link – the most fundamental part of HTML... `<a href="/categories/list">Back</a>`

**Documentation:** `ActionView::Helpers::UrlHelper`

**Partial**

Rails naming convention: a partial ‘foo’ will go in a file `app\views\'action’\_foo.rhtml` (note the initial underscore).

The scaffold uses the same code to process both the ‘edit’ and ‘new’ actions, so it puts the code into a partial, invoked by the `render_partial` method.

```html
app\views\categories\_form.rhtml
<%=
  error_messages_for 'category'
%>
<!-[form:category]->
<p><label for="category_category">Category</label><br/>
<%=
  text_field 'category', 'category'
%></p>
<p><label for="category_created_on">Created on</label><br/>
</p>
<p><label for="category_updated_on">Updated on</label><br/>
</p>
<!-[eoform:category]->
```

- `error_messages_for` returns a string with marked-up text for any error messages produced by a previous attempt to submit the form. If one or more errors is detected, the HTML looks like this:

```
<div class="errorExplanation" id="errorExplanation">
<h2>n errors prohibited this xxx from being saved</h2>
<p>There were problems with the following fields:</p>
<ul>
  <li>field_1 error_message_1</li>
  ... ...</ul>
<li>field_n error_message_n</li>
</div>
```

We saw this in action on Day 1 - *Illustration 2: Capturing data errors* on page 7. Note: the css tags match
corresponding statements in the stylesheet created by the generate scaffold script.

**Documentation: ActionView::Helpers::ActiveRecordHelper**

- `text_field` is a Rails Helper which generates this HTML:
  ```html
  <input id="category_category" name="category[category]" size="30" type="text" value="" />
  ```
  The first parameter is the table name; the second is the field name.

**Documentation: ActionView::Helpers::FormHelper**

Note a little bug in Rails – it knows not to create input fields for the reserved field names `created_on` and `updated_on`, but it still generates labels for them.

**The Rendered View for the “New” action**

We’re now in a position to look at the code that’s returned to the browser in response to the “New” action, and see where it’s all come from. The Layout supplies the **bold** text; the Template the **Regular** text; and the Partial the **Italic** text:

```html
app/views/categories/new.rhtml
<html>
<head>
  <title>Categories: new</title>
  <link href="/stylesheets/scaffold.css" media="screen" rel="Stylesheet" type="text/css" />
</head>
<body>
  <h1>New category</h1>
  <form action="/categories/create" method="post">
    <!--[form:category]-->
    <p><label for="category_category">Category</label><br/>
      <input id="category_category" name="category[category]" size="30" type="text" value="" />
    </p>
    <p><label for="category_created_on">Created on</label><br/>
    </p>
    <p><label for="category_updated_on">Updated on</label><br/>
    </p>
    <!--[eoform:category]-->
    <input name="submit" type="submit" value="Create" />
  </form>
  <a href="/categories/list">Back</a>
</body>
</html>
```

**Analysing the View for the ‘List’ action**

The ‘Edit’ and ‘Show’ views are similar to the ‘New’ view. ‘List’ contains a few new tricks. Remember how the controller ran the following piece of code before going off to render the ‘List’ template:

```ruby
@category_pages, @categories = paginate :category, :per_page => 10
```

`paginate` populates the `@categories` instance variable with sorted records from the Categories table, `:per_page` records at a time, and contains all the logic for next page / previous page etc. navigation. `@category_pages` is a `Paginator` instance. How these are used in the template is explained at the end of the following section.

**Documentation: ActionController::Pagination**
The template is as follows:

```html
app/views/categories/list.rhtml
<h1>Listing categories</h1>
<table>
  <tr>
    <% for column in Category.content_columns %>
      <th><%= column.human_name %></th>
    <% end %>
  </tr>
  <% for category in @categories %>
    <tr>
      <% for column in Category.content_columns %>
        <td><%= h category.send(column.name) %></td>
      <% end %>
      <td><%= link_to 'Show', :action => 'show', :id => category %></td>
      <td><%= link_to 'Edit', :action => 'edit', :id => category %></td>
      <td><%= link_to 'Destroy', {:action => 'destroy', :id => category}, :confirm => "Are you sure?" %></td>
    </tr>
  <% end %>
</table>
<%= link_to "Previous page", { :page => @category_pages.current.previous } if @category_pages.current.previous %>
<%= link_to "Next page", { :page => @category_pages.current.next } if @category_pages.current.next %>
<br />
<%= link_to 'New category', :action => 'new' %>

• content_columns returns an array of column objects excluding any ‘special’ columns (the primary id, all columns ending in ‘_id’ or ‘_count’, and columns used for single table inheritance)

Documentation: ActionController::Base

• human_name is a synonym for human_attribute_name, which transforms attribute key names into a more human format, such as ‘First name’ instead of ‘first_name’

Documentation: ActiveRecord::Base

• h automatically ‘escapes’ HTML code. One of the problems with allowing users to input data which is then displayed on the screen is that they could accidentally (or maliciously) type in code which could break the system when it was displayed. To guard against this, it is good practice to ‘HTML escape’ any data which has been provided by users. This means that e.g. `</table>` is rendered as `<table>` which is harmless. Rails makes this really simple – just add an ‘h’ as shown

• confirm is a useful optional parameter for the link_to helper – it generates a Javascript pop-up box which forces the user to confirm the Destroy before actioning the link:

Illustration 3: Javascript pop-up

4 For example, think what would happen if a user typed in “</table>” as a Category.
The paging logic takes a bit of unravelling. Ruby can use `if` as a modifier: 

```ruby
expression if boolean-expression
```

`expression` evaluates `expression` only if `boolean-expression` is `true`. `@category_pages.current` returns a Page object representing the paginator's current page. `@category_pages.current.previous` returns a new Page object representing the page just before this page, or `nil` if this is the first page.

So, if there is a previous page to navigate to, then this construct will display a link; if there isn't, the link is suppressed.

The rendered code for page `n` will look like:

```html
<a href="/categories/list?page=[n-1]">Previous page</a>
<a href="/categories/list?page=[n+1]">Next page</a>
```

### Tailoring the Generated Scaffold Code

The code generated by the Scaffold script is perfectly usable ‘out of the box’, and is robust once you have added enough validation into your data model. However, if that’s all there was to developing Rails applications, then programmers would be out of a job, which would clearly not be a good thing :-) So let’s do some tailoring:

#### The Controller

In a ‘List’ view, I would expect the records to be displayed in alphabetical order. This requires a minor change to the controller:

```ruby
app/controllers/categories_controller.rb (excerpt)
def list
  @category_pages, @categories = paginate :category,
    :per_page => 10, :order_by => 'category'
end
```

The `flash` message will be picked up and displayed on the next screen to be displayed – in this case, the `list` screen. By default, the scaffold script doesn’t display flash messages - we’ll change this in a minute – see below.

#### The View

### Displaying Flash Messages

Rails provides a technique for passing ‘flash’ messages back to the user – e.g. an ‘Update Successful’ message which displays on the next screen and then disappears. These can be picked up easily with a small change to the Layout (adding it to the Layout means it will appear on any screen):
A simple addition to the stylesheet makes the flash message more conspicuous:

```html
.public\stylesheets\scaffold.css (excerpt)
.notice {
  color: red;
}
```

### Sharing Variables between the Template and Layout

Note that I've moved the `<h1>`...`</h1>` heading text out of the Template into the Layout so that it appears above the flash message. As each template will have a different heading, I need to set the value of the variable `@heading` in the Template. Rails is quite ok with this – Template variables are available to Layouts at rendering time.

I've made this change and some formatting changes to come up with my finished template:

```html
app\views\categories\list.rhtml
<% @heading = "Categories" %>
<table>
  <tr>
    <th>Category</th>
    <th>Created</th>
    <th>Updated</th>
  </tr>
  <% for category in @categories %>
    <tr>
      <td><%=h category["category"] %></td>
      <td><%= category["created_on"].strftime("%I:%M %p %d-%b-%y") %></td>
      <td><%= category["updated_on"].strftime("%I:%M %p %d-%b-%y") %></td>
      <td><%= link_to 'Edit', :action => 'edit', :id => category %></td>
      <td><%= link_to 'Delete', {:action => 'destroy', :id => category}, :confirm => "Are you sure you want to delete this category?" %></td>
    </tr>
  <% end %>
</table>
<br />
<%= link_to 'New category', :action => 'new' %>
<% if @category_pages.page_count>1 %>
  hr />
Page: <%=pagination_links @category_pages %>
<hr />
<% end %>
<% end %>
```

- I don’t like the default date format, so I use a Ruby method `strftime()` to format the date and time fields the way I want them.
Tidying up the Edit and New Screens

A few changes to the Partial used by 'New' and 'Edit': use a table to improve the layout; get rid of the unwanted
created_on/updated_on labels; and prevent the user typing too much into the Category field:

```ruby
<%= error_messages_for 'category' %>
<table>
  <tr>
    <td><b><label for="category_category">Category:</label></b></td>
    <td><%= text_field "category", "category", "size"=>20, "maxlength"=>20 %></td>
  </tr>
</table>
```

and a few minor changes to the two templates (note in particular the use of @heading):

```ruby
<% @heading = "Edit Category" %>
<%= start_form_tag :action => 'update', :id => @category %>
  <%= render_partial "form" %>
  <%= submit_tag "Save" %>
<%= end_form_tag %>
<%= link_to 'Back', :action => 'list' %>

<% @heading = "New Category" %>
<%= start_form_tag :action => 'create' %>
  <%= render_partial "form" %>
  <%= submit_tag "Save" %>
<%= end_form_tag %>
<%= link_to 'Back', :action => 'list' %>
```

That takes us to the end of Day 2. We have a working system for maintaining our Categories table, and have started to take control of the scaffold code which Rails has generated.
Day 3 on Rails

Now it’s time to start on the heart of the application. The Items table contains the list of ‘To Dos’. Every Item may belong to one of the Categories we created on Day 2. An Item optionally may have one Note, held in a separate table, which we will look at tomorrow. Each table has a primary key ‘id’, which is also used to record links between the tables.

![Diagram of data model](attachment:categories-items-notes.png)

Illustration 4: Simplified Data Model

The ‘Items’ Table

**MySQL table definition**

The fields in the Items table are as follows:

- done - 1 means the To Do item has been completed
- priority – 1 (high priority) to 5 (low priority)
- description – free text stating what is to be done
- due_date – stating when it is to be done by
- category_id – a link to the Category this item comes under (‘id’ in the Categories table)
- note_id – a link to an optional Note explaining this item (‘id’ in the Notes table)
- private – 1 means the To Do item is classed as ‘Private’

```sql
CREATE TABLE items (
    id smallint(5) unsigned NOT NULL auto_increment,
    done tinyint(1) unsigned NOT NULL default '0',
    priority tinyint(1) unsigned NOT NULL default '3',
    description varchar(40) NOT NULL default '',
    due_date date default NULL,
    category_id smallint(5) unsigned NOT NULL default '0',
    note_id smallint(5) unsigned default NULL,
    private tinyint(3) unsigned NOT NULL default '0',
    created_on timestamp(14) NOT NULL,
    updated_on timestamp(14) NOT NULL,
    PRIMARY KEY (id)
) TYPE=MyISAM COMMENT='List of items to be done';
```

The Model

As before, Rails can generate an empty model file:

```
W:\ToDo>ruby script/generate model item
exists  app/models/
exists  test/unit/
exists  test/fixtures/
create  app/models/item.rb
create  test/unit/item_test.rb
create  test/fixtures/items.yml
W:\ToDo>
```

5 MySQL doesn’t have a ‘boolean’ type, so we have to use 0/1
which we can populate:

```ruby
app/models/item.rb

class Item < ActiveRecord::Base
  belongs_to :category
  validates_associated :category
  validates_format_of :done_before_type_cast, :with => /[01]/, :message=>"must be 0 or 1"
  validates_inclusion_of :priority, :in=>1..5, :message=>"must be between 1 (high) and 5 (low)"
  validates_presence_of :description
  validates_length_of :description, :maximum=>40
  validates_format_of :private_before_type_cast, :with => /[01]/, :message=>"must be 0 or 1"
end
```

**Validating Links between Tables**

- the use of `belongs_to` and `validates_associated` links the Items table with the `item_id` field in the Category table.

**Documentation:** ActiveRecord::Associations::ClassMethods

**Validating User Input**

- `validates_presence_of` protects ‘NOT NULL’ fields against missing user input
- `validates_format_of` uses regular expressions to check the format of user input
- when a user types input for a numeric field, Rails will always convert it to a number – if all else fails, a zero. If you want to check that the user has actually typed in a number, then you need to validate the input _before_type_cast_, which lets you access the ‘raw’ input.
- `validates_inclusion_of` checks user input against a range of permitted values
- `validates_length_of` prevents the user entering data which would be truncated when stored.

**Documentation:** ActiveRecord::Validations::ClassMethods

**The ‘Notes’ table**

This table contains a single free text field to hold further information for a particular To Do Item. This data could of course have been held in a field on the Items table; however, if you do it this way you’ll learn a lot more about Rails :)  

**MySQL table definition**

```sql
Notes table

CREATE TABLE notes (  
id smallint(6) NOT NULL auto_increment,  
more_notes text NOT NULL,  
updated_on timestamp(14) NOT NULL,  
PRIMARY KEY (id)  
) TYPE=MyISAM COMMENT='Additional optional information for to-dos';
```

**The Model**

Generate the empty model file, but it contains nothing new:

```ruby
app/models/note.rb

class Note < ActiveRecord::Base  
  validates_presence_of :more_notes
end
```

---

6 What might seem a more obvious alternative: `validates_inclusion_of :done before_type_cast, :in=>"0".."1", :message=>"must be between 0 and 1"` – fails if the input field is left blank
7 You could combine the two rules for the Description field into one: `validates_length_of :description, :within => 1..40`
but we need to remember to add this link into the Items model:

**app/models/item.rb (excerpt)**

```ruby
class Item < ActiveRecord::Base
  belongs_to :note
end
```

### Using a Model to maintain Referential Integrity

The code we are about to develop will allow a user to add one Note to any Item. But what happens when a user deletes an Item which has an associated Note? Clearly, we need to find a way of deleting the Note record too, otherwise we get left with ‘orphaned’ Notes records.

In the Model / View / Controller way of doing things, this code belongs in the Model. Why? well, you'll see later that we can delete Item records by clicking on a Dustbin icon on the ‘To Do’ screen, but we can also delete them by clicking on Purge completed items. By putting the code into the Model, it will be run regardless of where the delete action comes from.

**app/models/item.rb (excerpt)**

```ruby
def before_destroy
  unless note_id.nil?
    Note.find(note_id).destroy
  end
end
```

This reads: before you delete an Item record, find the record in Notes whose id equals the value of Note_id in the Item record you are about to delete, and delete it first. Unless there isn't one :-)

Similarly, if a record is deleted from the Notes table, then any reference to it in the Items table needs to be erased:

**app/models/note.rb (excerpt)**

```ruby
def before_destroy
  Item.find_by_note_id(id).update_attribute('note_id', NIL)
end
```

**Documentation: ActiveRecord::Callbacks**

### More Scaffolding

Let’s generate some more scaffold code. We’ll do this for both the Items table and the Notes table. We aren’t ready to work on Notes as yet, but having the scaffold in place means we can refer to Notes in today’s coding without generating lots of errors. Just like building a house – scaffolding allows you to build one wall at a time without everything crashing around your ears.

```
W:\ToDo>ruby script/generate scaffold Item
[snip]
W:\ToDo>ruby script/generate scaffold Note
[snip]
W:\ToDo>
```

Note: as we tailored the stylesheet yesterday, reply “n” to the “overwrite public/stylesheets/scaffold.css? [Ynaq]” prompt.

### More on Views

**Creating a Layout for the Application**

By now, it is becoming obvious that all my templates will have the same first few lines of code, so it makes sense to move this common code into an application-wide layout. Delete all the app/views/layouts/*.rhtml files,
and replace with a common application.rhtml.

```html
app/views/layouts/application.rhtml

<html>
<head>
  <title><%= @heading %></title>
  <%= stylesheet_link_tag 'todo' %>
  <script language="JavaScript">
    <!-- Begin
    function setFocus() {
      if (document.forms.length > 0) {
        var field = document.forms[0];
        for (i = 0; i < field.length; i++) {
          if ((field.elements[i].type == "text") || (field.elements[i].type == "textarea") || (field.elements[i].type.toString().charAt(0) == "s")) {
            document.forms[0].elements[i].focus();
            break;
          }
        }
      }
    }
    // End -->
  </script>
</head>
<body OnLoad="setFocus()">
<h1><%= @heading %></h1>
<% if @flash["notice"] %>
  <span class="notice">
    <%=h @flash["notice"] %>
  </span>
<% end %>
<%= @content_for_layout %>
</body>
</html>
```

The @heading set in the Template is now used for the <title> as well as <h1>. I’ve renamed the public/stylesheets/scaffold.css to todo.css for tidiness, and also generally played with colours, table borders, to give a prettier layout. I’ve also added in a little Javascript to automatically position the cursor in the first input field in the browser ready for the user to start typing.

The ‘To Do List’ screen

What I’m trying to achieve is a look based on a PalmPilot or similar PDA desktop. The end product is shown in Illustration 5: The ‘To Do List’ Screen.

Some points:

• clicking on the ‘tick’ (✓) column heading will purge all the completed items (those marked with a tick)
• the display can be sorted by clicking on the ‘Pri’, ‘Description’, ‘Due Date’, and ‘Category’ column headings
• the 0/1 values for ‘Done’ are converted into a little ‘tick’ icon
• items past their due date are coloured red and shown in bold
• the presence of an associated note is shown by ‘note’ icon
• the 0/1 values for ‘Private’ are converted into a padlock symbol
• individual items can be edited or deleted by clicking on the icons on the right of the screen
• the display has a nice ‘striped’ effect
• new items can be added by clicking on the ‘New To Do...’ button at the bottom of the screen
• there’s a button link to the ‘Categories’ stuff from day 2

It’s amazing what a few lines in a stylesheet can do to change the appearance of a screen, plus of course a collection of icons...

Page 22
The template used to achieve this is built up as follows:

```ruby
app/views/items/list.rhtml
<% @heading = "To Do List" %>
<%= start_form_tag :action => 'new' %>
<table>
<tr>
<th><%= link_to_image "done", {:action => "purge_completed"}, :confirm => "Are you sure you want to permanently delete all completed To Dos?" %></th>
<th><%= link_to_image "priority",{:action => "list_by_priority"}, "alt" => "Sort by Priority" %></th>
<th><%= link_to_image "description",{:action => "list_by_description"}, "alt" => "Sort by Description" %></th>
<th><%= link_to_image "due_date",{:action => "list"}, "alt" => "Sort by Due Date" %></th>
<th><%= link_to_image "category",{:action => "list_by_category"}, "alt" => "Sort by Category" %></th>
<th><%= show_image "note" %></th>
<th><%= show_image "private" %></th>
</tr>
<% render_collection_of_partials "list_stripes", @items %>
</table>
<hr />
<%= submit_tag "New To Do..." %>
<%= submit_tag "Categories...", {:type => 'button', :onClick=>"parent.location='" + url_for( :controller => 'categories', :action => 'list' ) + '"" + '1" %>
<% end_form_tag %>
<%= "Page: " + pagination_links(@item_pages, :params => { :action => @params['action'] || "index" }) + "\r\n" if @item_pages.page_count>1 %>
```

**Purging completed ‘To Dos’ by clicking on an icon**

Clickable images are created by `link_to_image`, which by default expects to find an image in `pub/images` with a `.png` suffix; clicking on the image will run the specified method.

Adding in the :confirm parameter generates a javascript pop-up dialogue box as before.

*Documentation: ActionView::Helpers::UrlHelper*
Clicking ‘OK’ will invokes the **purge_completed** method. This new **purge_completed** method needs to be defined in the controller:

```ruby
app\controllers\items_controller.rb (excerpt)
def purge_completed
  Item.destroy_all "done = 1"
  redirect_to :action => 'list'
end
```

**Item.destroy_all** deletes all the records in the **Items** table where the value of the field **done** is 1, and then reruns the list action.

---

**Changing the Sort Order by clicking on the Column Headings**

Clicking on the Pri icon invokes a **list_by_priority** method. This new **list_by_priority** method needs to be defined in the controller:

```ruby
app\controllers\items_controller.rb (excerpt)
def list
  @item_pages, @items = paginate :item,
  :per_page => 10, :order_by => 'due_date,priority'
end
```

```ruby
def list_by_priority
  @item_pages, @items = paginate :item,
  :per_page => 10, :order_by => 'priority,due_date'
  render_action 'list'
end
```

We’ve specified a sort order for the default list method, and created a new **list_by_priority** method. Note also that we need to explicitly **render_action 'list'**, as by default Rails would try to render a template called **list_by_priority** (which doesn’t exist :-)

---

**Adding a Helper**

The headings for the Note and Private columns are images, but are not clickable. I decided to write a little method **show_image(name)** to just show the image:

```ruby
app\helpers\application_helper.rb
module ApplicationHelper
  def self.append_features(controller)
    controller.ancestors.include?(ActionController::Base) ?
      controller.add_template_helper(self) : super
  end

  def show_image(src)
    img_options = { "src" => src.include?("/") ? src : "/images/#{src}" }
    img_options["src"] = img_options["src"] + ".png" unless
    img_options.include?(".*")
    img_options["border"] = "0"
    tag("img", img_options)
  end
end
```

Once this helper has been linked in by the controller:

```ruby
app\controllers\application.rb
class ApplicationController < ActionController::Base
  helper :Application
end
```

---

9 list by description and list by category are similar and are left as an easy exercise for the reader. However, if you get stuck with list by category, see Still to be done on page 39
it is available for all the templates in the application.

**Using Javascript Navigation Buttons**

`onClick` is a standard Javascript technique for handling button actions such as navigating to a new web page. However, Rails goes to great lengths to rewrite pretty URLs, so we need to ask Rails for the correct URL to use. Given a controller and an action, `url_for` will return the URL.

**Formatting a Table with a Partial**

I wanted to create a nice stripey effect for the list of items. **Partials** provide the solution; they can either be invoked by the `render_partial` method:

```ruby
<% for item in @items %
  <%= render_partial "list_stripes", item %>
<% end %>
```

or by the more economical `render_collection_of_partials`:

```ruby
render_collection_of_partials "list_stripes", @items
```

Rails also passes a sequential number `list_stripes_counter` to the Partial. This is the key to formatting alternate rows in the table with either a light grey background or a dark grey background. One way is simply to test whether the counter is odd or even: if odd, use light gray; if even, use dark gray.

The completed Partial is as follows:

```ruby
app\views\items\_list_stripes.rhtml

<tr class="<%= list_stripes_counter.modulo(2).nonzero? ? "dk_gray" : "lt_gray" %>">
  <td style="text-align: center"><%= list_stripes["done"] == 1 ? show_image("done_ico.gif") : " " %></td>
  <td style="text-align: center"><%= list_stripes["priority"] %></td>
  <td><%=h list_stripes["description"] %></td>
  <td>&nbsp;</td>
  <td>&nbsp;</td>
  <td><%=h list_stripes.category ? list_stripes.category["category"] : "Unfiled" %></td>
  <td><%= link_to_image("edit", { :controller => 'items', :action => "edit", :id => list_stripes.id }) %></td>
  <td><%= link_to_image("delete", { :controller => 'items', :action => "destroy", :id => list_stripes.id }, :confirm => "Are you sure you want to delete this item?") %></td>
</tr>
```

A little bit of Ruby is used to test if the counter is odd or even and render either `class="dk_gray"` or `class="lt_gray"`:

```
list_stripes_counter.modulo(2).nonzero? ? "dk_gray" : "lt_gray"
```

the code as far as the first question mark asks: *is the remainder when you divide list_stripes_counter by 2 nonzero?*

**Ruby Documentation: class Numeric**

The remainder of the line is actually a cryptic *if then else* expression which sacrifices readability for brevity: *if the
expression before the question mark is true, return the value before the colon; else return the value after the colon.

Ruby Documentation: Expressions

The two tags `dk_gray` and `lt_gray` are then defined in the stylesheet:

```css
.public\stylesheets\ToDo.css (excerpt)
.lt_gray { background-color: #e7e7e7; }
.dk_gray { background-color: #d6d7d6; }
```

Note: the same `if then else` construct is used to display the 'tick' icon if `list_stripes['done']` equals 1, otherwise display an HTML blank space character:

```ruby
list_stripes['done'] == 1 ? show_image('done_ico') : "\n"
```

Formatting based on Data Values

It's also easy to highlight specific data items – for example, dates in the past.

```ruby
list_stripes['due_date'] < Date.today ? '<td class="past_due">' : '<td>'
```

Again, this needs a matching `.past_due` stylesheet entry.

Handling Missing Values in a Lookup

We want the system to be able to cope with the situation where the user deletes a Category which is in use by To Do items. In this case, the Category should be displayed as 'Unfiled':

```ruby
list_stripes.category ? list_stripes.category['category'] : 'Unfiled'
```

OK. if you’ve followed this so far, you should have a 'To Do List' screen looking something like Illustration 5

The 'To Do List' Screen

Turning next to what happens when the ‘New To Do...’ button is pressed. Again, there are few new tricks lurking in the code.

The template is minimal:

```ruby
<% @heading = "New To Do" %>
<%= error_messages_for 'item' %>
<%= start_form_tag :action => 'create' %>
```

New To Do

<table>
<thead>
<tr>
<th>Description:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Date due:</td>
<td>2005</td>
</tr>
<tr>
<td>Category:</td>
<td>Home and Family</td>
</tr>
<tr>
<td>Priority:</td>
<td>3</td>
</tr>
<tr>
<td>Private?</td>
<td></td>
</tr>
<tr>
<td>Complete?</td>
<td></td>
</tr>
</tbody>
</table>

Illustration 6 New 'To Do' screen
and the real work is done in the partial, where it can be shared with the ‘Edit’ action:

```
<app\views\items\ form.rhtml>
<tr>
  <td>Description: </td>
  <td><%= text_field "item", "description", "size" => 40, "maxlength" => 40 %></td>
</tr>
<tr>
  <td>Due date: </td>
  <td><%= date_select "item", "due_date", :use_month_numbers => true %></td>
</tr>
<tr>
  <td>Category: </td>
  <td><%= select "item","category",[1,2,3,4,5] %></td>
</tr>
<tr>
  <td>Priority: </td>
  <td><%= select "item","priority",[1,2,3,4,5] %></td>
</tr>
<tr>
  <td>Private? </td>
  <td><%= check_box "item","private" %></td>
</tr>
<tr>
  <td>Complete? </td>
  <td><%= check_box "item", "done" %></td>
</tr>
</table>
```

Creating a Drop-down List for a Date Field

date_select generates a rudimentary drop-down menu for date input:

```
date_select "item", "due_date", :use_month_numbers => true
```

Documentation: ActionView::Helpers::DateHelper

Trapping Exceptions in Ruby

Unfortunately, date_select quite happily accepts dates like 31st February. Rails then dies when it tries to save this ‘date’ to the database. One workaround is to trap this failed save using rescue, a Ruby exception handling method.
Redirect to : action => 'new'
end

Ruby Documentation: Exceptions, Catch, and Throw

Creating a Drop-down List from a Lookup Table

This is another example of Rails solving an everyday coding problem in an extremely economical way. In this example:

```ruby
options_from_collection_for_select @categories, "id", "category", @item.category_id
```

`options_from_collection_for_select` reads all the records in categories and renders them as `<option value="[value of id]">[value of category]</option>`. The record that matches `@item.category_id` will be tagged as 'selected'. As is this wasn't enough, the code even html_escapes the data for you. Neat.

Documentation: ActionView::Helpers::FormOptionsHelper

Note that data driven drop down boxes have to get their data from somewhere – which means an addition to the controller:

```ruby
app/controllers/items_controller.rb (excerpt)
def new
  @categories = Category.find_all
  @item = Item.new
end
def edit
  @categories = Category.find_all
  @item = Item.find(@params[:id])
end
```

Creating a Drop-down List from a List of Constants

This is a simpler version of the previous scenario. Hard-coding lists of values into selection boxes isn't always a good idea – it's easier to change data in tables than edit values in code. However, there are cases where it's a perfectly valid approach, so in Rails you do:

```ruby
select "item","priority",[1,2,3,4,5]
```

Note also how to set a default value in the previous line of code.

Documentation: ActionView::Helpers::FormOptionsHelper

Creating a Checkbox

Another regular requirement; another helper in Rails:

```ruby
check_box "item","private"
```

Documentation: ActionView::Helpers::FormHelper

Finishing Touches

Tailoring the Stylesheet

At this point, the ‘To Do List’ screen should work, and so should the ‘New To Do’ button. To produce the screens shown here, I also made the following changes to the stylesheet:

```css
body {
  background-color: #c6c3c6;
  color: #333;
}

.notice {
  color: red;
  background-color: white;
}
```
The ‘Edit To Do’ Screen

The rest of Day 3 is taken up building the ‘Edit To Do’ screen, which is very similar to the ‘New To Do’. I used to get really annoyed with college text books which stated: *this is left as an easy exercise for the reader*, so now it’s great to be able to do the same to you\(^\text{10}\).

Which takes us to the end of Day 3 – and the application now looks nothing like a Rails scaffold, but under the surface, we’re still using a whole range of Rails tools to make development easy.

\(^{10}\) But unlike my college text book authors, I do reveal the answers on Day 4 :-) - see *app/views/items/edit.rhtml* on page 31
Day 4 on Rails

The ‘Notes’ screens

Linking ‘Notes’ to the ‘Edit To Do’

Although the Notes scaffold code gives the full CRUD facilities, we don’t want the user to invoke any of this directly. Instead, if an Item has no associated Note, we want to be able to create one by clicking on a Notes icon on the Edit To Do screen:

![Illustration 7: Creating a New Note from the ‘Edit To Do’ screen](image)

If a Note already exists, we want to edit or delete it by clicking on the appropriate icon on the Edit To Do screen:

![Illustration 8: Editing or Deleting an existing Note](image)

First of all, let’s look at the code for the ‘Edit To Do’ screen. Note how the Notes buttons change according to whether a Note already exists, and how control is transferred to the Notes controller:

```ruby
class ItemsController < ApplicationController
  def edit
    @item = Item.find(params[:id])
    Note.find_or_create_for_item(@item)
    # code for rendering the 'edit' form
  end
end
```

```ruby
app/views/items/edit.rhtml
<% @heading = "Edit To Do" %>
<%= error_messages_for 'item' %>
<%= start_form_tag :action => 'update', :id => @item %>
<table>
  <%= render_partial "form" %>
</table>
The 'Edit Notes' Screen

Editing an existing Note is pretty straightforward. This is the Template:

```
app/views/notes/edit.rhtml

```%
@heading = "Edit Note"
```
<u>
start_form_tag :action => 'update', :id => @note
render_partial "form"
```
submit_tag "Save"
```
submit_tag "Cancel", {:type => 'button', :onClick=>"parent.location='" + url_for( :controller => 'items', :action => 'list' ) + "'" } %>
```
end_form_tag %>
```

and its matching Partial:

```
app/views/notes_form.rhtml

```<table>
<tr>
<td><label for="note_more_notes">More notes</label></td>
<td><%= text_area 'note', 'more_notes'  %></td>
</tr>
</table>
```

Once the update or destroy of the Notes table is complete, we want to return to the 'To Do List' screen:

```
app/controllers/notes_controller.rb (excerpt)

def update
  @note = Note.find(@params[:id])
  if @note.update_attributes(@params[:note])
    flash['notice'] = 'Note was successfully updated.'
    redirect_to :controller => 'items', :action => 'list'
  else
    render_action 'edit'
  end
end

def destroy
  Note.find(@params[:id]).destroy
  redirect_to :controller => 'items', :action => 'list'
end
```

Remember that the referential integrity rules we have already created will ensure that when a Note is deleted, any references to it in Items will be removed too (see Using a Model to maintain Referential Integrity on page 21).

The 'New Note' Screen

Create is a bit more tricky. What we want to do is:
• store the new note in the Notes table
• find the id of the newly created record in the Notes table
• record this id back in the notes_id field of the associated record in the Items table

Session variables provide a useful way of persisting data between screens – we can use them here to store the Id of the record in the Notes table.

**Documentation: ActionController::Base**

### Saving and retrieving Data using Session Variables

First of all, when we go off to create the new Notes record, we pass the id of the Item we are editing:

```ruby
<%= link_to_image "note", :controller => "notes", :action => "new", :id => @item.id %></td>
```

The `new` method in the Notes controller stores this away in a session variable:

```ruby
def new
  @session[:item_id] = @params[:id]
  @note = Note.new
end
```

The ‘New Notes’ template has no surprises:

```ruby
<%= @heading = "New Note" %><%= start_form_tag :action => 'create' %><%= render_partial "form" %><%= submit_tag "Save" %><%= submit_tag "Cancel", {:type => 'button', :onClick=>"parent.location='" + url_for(:controller => 'items', :action => 'list') + "'" } %><%= end_form_tag %>
```

The `create` method retrieves the session variable again and uses it to find the record in the Items table. It then updates the note_id in the Item table with the id of the record it has just created in the Note table, and returns to the Items controller again:

```ruby
def create
  @note = Note.new(@params[:note])
  if @note.save
    flash['notice'] = 'Note was successfully created.'
    @item = Item.find(@session[:item_id])
    @item.update_attribute(:note_id, @note.id)
    redirect_to :controller => 'items', :action => 'list'
  else
    render_action 'new'
  end
end
```

### Changing the ‘Categories’ Screens

There isn’t a great deal left to do on the system now, other than tidy up the templates created in earlier days so they have the same style of navigation buttons:

```ruby
<%= @heading = "Categories" %><%= form action="/categories/new" method="post"%><table><tr><th>Category</th>
```
Navigation through the system

The final navigation paths through the application are shown below. Any redundant scaffold code – e.g. the `show.rhtml` files – can be simply deleted. That's the beauty of scaffold code – it didn't cost you any effort to code it in the first place, and once it's served its purpose, just get rid of it.
Setting the Home Page for the Application

As a final step, we need to kill the default 'Welcome to Rails' screen if the user points their browser to http://todo. There are two steps:

- Add the home page definition to the Routes file:

```
config/routes.rb (excerpt)
map.connect '', :controller => 'items'
```

- rename `public\index.html` to `public\index.html.orig`

Downloading a Copy of this Application

If you’d like a copy of the ‘To Do’ application to play with, there’s a link on http://rails.homelinux.org. You’ll need to:

- use Rails to set up the directory structure (see Running the Rails script on page 3)
- download the todo_app.zip file into the newly created ToDo directory
- unzip the files `unzip -o todo_app.zip`
- rename `public\index.html` to `public\index.html.orig`
- if you want to use the sample database, `mysql -uroot -p < db/ToDo.sql`

and finally

I hope you found this document useful – I’m always happy to receive feedback, good or bad, to jpmcc@users.sourceforge.net.

Happy coding with Rails!
Appendix – afterthoughts

After writing ‘Four Days’, I got a huge amount of feedback which greatly helped improve the quality of the document. One question did crop up repeatedly - “how do you update more than one record from the same screen” - so here’s an appendix covering this most Frequently Asked Question. It isn’t the easiest Rails concept to grasp, and it’s an area I would expect to see more “Helpers” appearing in the future.

Multiple Updates

In the screenshot below, the user can tick/untick multiple “To Dos” using the checkboxes in the extreme left hand column, and then press “Save” to store the results in the database.

<table>
<thead>
<tr>
<th>To Do List</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Rails supports multiple updates with another naming convention, which is to append the id of the record you are editing to the name within square brackets []. This enables you to pick out a particular record from multiple records on the screen.

Let’s work backwards from the HTML we are trying to generate. This is what it looks like for a record with id = 6:

```html
<td style="text-align: center">
<input type="checkbox" id="item_done" name="item[6][done]" value="1" checked />
<input name="item[6][done]" type="hidden" value="0" />
</td>
```

(“checked” is omitted if the checkbox is not checked)

One way to generate this code is:

```ruby
app/view/items/_list_stripes.rhtm (excerpt)
<td style="text-align: center">
   <%= check_box_tag("item["+list_stripes.id.to_s+"].done","1",list_stripes["done"]==1) %>
</td>
```
The parameters for `check_box_tag` are `name`, `value = "1"`, `checked = false`, `options = {}`; for `hidden_field_tag` `name`, `value = nil`, `options = {}`

Documentation: ActionView::Helpers::FormTagHelper

Plus of course we now need a Save button:

```erb
app/views/items/list.rhtml (excerpt)
<% @heading = "To Do List" %>
<%= start_form_tag :action => 'updater' %>
<table>
  ...
</table>
<hr />
<%= submit_tag "Save" %>
<%= submit_tag "New To Do...", {:type => 'button', :onClick => "parent.location='" + url_for( :controller => 'items', :action => 'new' ) + "'"} %>
<%= submit_tag "Categories...", {:type => 'button', :onClick => "parent.location='" + url_for( :controller => 'categories', :action => 'list' ) + "'"} %>
<%= end_form_tag %>
<%= "Page: " + pagination_links(@item_pages, :params => { :action => @params['action'] || "index" }) + "<hr />" if @item_pages.page_count>1 %>
```

Controller

What gets returned to the controller when you press the ‘Save’ button is the following hash:

```ruby
params: {
  :controller=>"items",
  :item=> {
    "6"=>{"done"=>"0"},
    ... etc...
    "5"=>{"done"=>"1"}
  },
  :action=>"updater"
}
```

We’re interested in the `:item` bit. For example, the bold line means “the record with `id = 6` has the value of the `done` field set to `0`”. From here, it’s a fairly easy job to update the `Items` table:

```ruby
app\controller\items_controller (excerpt)
def updater
  @params[:item].each { |item_id, attr|
    item = Item.find(item_id)
    item.update_attribute(:done, attr[:done])
  }
  redirect_to :action => 'list'
end
```

Each puts “6” into the variable `item_id`, and “done” => “0” into `attr`.

This code works, but if you watch what is happening in `development.log`, you’ll see that Rails is retrieving and updating every record, whether it’s changed or not. Not only is this creating unnecessary database updates, but it also means that `updated_on` also gets changed, which isn’t really what we want. Much better to only update if ‘done’ has changed, but this means some coding -:-

```ruby
app\controller\items_controller (excerpt)
def updater
  @params[:item].each { |item_id, contents|
    item = Item.find(item_id)
    if item.done != contents[:done].to_i
```
Note that we need to convert the string done to an integer using to_i so we can compare like with like. This is the kind of gotcha you can easily miss – it’s worth checking development.log from time to time to make sure Rails is doing what you expect.

**User Interface considerations**

This code works, and could be applied to make any field on the screen editable (another easy exercise for the reader :-). It does raise some interesting questions about what the user would expect. What if the user changes some check boxes, and then presses “New To Do...”, or re-sorts the display, without pressing “Save”? Should the system always “Save” before doing any other action? More easy exercises for the reader...

**Still to be done**

On page 24 I left list_by_category as an easy exercise for the reader. It proved to be less easy than it looked – in fact, I’m still looking for an elegant ‘Rails’ way to sort by a field in a lookup table. I ended up with this rather horrible code:

```ruby
app\controller\items_controller (excerpt)
def list_by_category
  @item_pages = Paginator.new self, Item.count, 10, @params['page']
  @items = Item.find_by_sql 'SELECT i.*, c.category FROM categories c, items i ' +
    'WHERE ( c.id = i.category_id ) '+
    'ORDER BY c.category '+
    'LIMIT 10 '+
  "OFFSET #{@item_pages.current.to_sql[1]}"
  render_action 'list'
end
```

If anyone has a better solution, please let me know. I leave this code as a reassuring example that if all else fails, Rails will not leave you stuck but will allow you to resort to ‘old-fashioned’ coding!

Enjoy coding with Rails!
## Index of Rails and Ruby Terms used in this Document

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
</tr>
<tr>
<td>action_name</td>
<td>11</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
</tr>
<tr>
<td>before_type_cast</td>
<td>20</td>
</tr>
<tr>
<td>belongs_to</td>
<td>20</td>
</tr>
<tr>
<td>C</td>
<td>14</td>
</tr>
<tr>
<td>check_box</td>
<td>28</td>
</tr>
<tr>
<td>check_box_tag</td>
<td>38</td>
</tr>
<tr>
<td>confirm</td>
<td>14, 23</td>
</tr>
<tr>
<td>content_columns</td>
<td>14</td>
</tr>
<tr>
<td>content_for_layout</td>
<td>11</td>
</tr>
<tr>
<td>created_at</td>
<td>5</td>
</tr>
<tr>
<td>created_on</td>
<td>5, 13</td>
</tr>
<tr>
<td>current</td>
<td>15</td>
</tr>
<tr>
<td>D</td>
<td>27</td>
</tr>
<tr>
<td>date_select</td>
<td>27</td>
</tr>
<tr>
<td>destroy</td>
<td>10</td>
</tr>
<tr>
<td>destroy_all</td>
<td>24</td>
</tr>
<tr>
<td>development.log</td>
<td>3, 10, 39</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
</tr>
<tr>
<td>end_form_tag</td>
<td>12</td>
</tr>
<tr>
<td>error_messages_for</td>
<td>12</td>
</tr>
<tr>
<td>F</td>
<td>10</td>
</tr>
<tr>
<td>find</td>
<td>10</td>
</tr>
<tr>
<td>find_all</td>
<td>10</td>
</tr>
<tr>
<td>Flash</td>
<td>15</td>
</tr>
<tr>
<td>H</td>
<td>14</td>
</tr>
<tr>
<td>h</td>
<td>14</td>
</tr>
<tr>
<td>helper</td>
<td>11</td>
</tr>
<tr>
<td>hidden_field_tag</td>
<td>38</td>
</tr>
<tr>
<td>HTML escape</td>
<td>14</td>
</tr>
<tr>
<td>human_attribute_name</td>
<td>14</td>
</tr>
<tr>
<td>human_name</td>
<td>14</td>
</tr>
<tr>
<td>I</td>
<td>5</td>
</tr>
<tr>
<td>id</td>
<td>5</td>
</tr>
<tr>
<td>L</td>
<td>11</td>
</tr>
<tr>
<td>Layout</td>
<td>11</td>
</tr>
<tr>
<td>link_to</td>
<td>12</td>
</tr>
<tr>
<td>link_to_image</td>
<td>23</td>
</tr>
<tr>
<td>lock_version</td>
<td>5</td>
</tr>
<tr>
<td>N</td>
<td>10</td>
</tr>
<tr>
<td>new</td>
<td>10</td>
</tr>
<tr>
<td>options_from_collection_for_select</td>
<td>28</td>
</tr>
<tr>
<td>P</td>
<td>13</td>
</tr>
<tr>
<td>paginate</td>
<td>13</td>
</tr>
<tr>
<td>pagination_links</td>
<td>17</td>
</tr>
<tr>
<td>Partial</td>
<td>11</td>
</tr>
<tr>
<td>previous</td>
<td>15</td>
</tr>
<tr>
<td>R</td>
<td>10</td>
</tr>
<tr>
<td>redirect_to</td>
<td>10</td>
</tr>
<tr>
<td>Referential Integrity</td>
<td>21</td>
</tr>
<tr>
<td>render_collection_ofpartials</td>
<td>25</td>
</tr>
<tr>
<td>render_partial</td>
<td>12, 25</td>
</tr>
<tr>
<td>render_template</td>
<td>10</td>
</tr>
<tr>
<td>rescue</td>
<td>27</td>
</tr>
<tr>
<td>S</td>
<td>10</td>
</tr>
<tr>
<td>save</td>
<td>10</td>
</tr>
<tr>
<td>select</td>
<td>28</td>
</tr>
<tr>
<td>session_variable</td>
<td>33</td>
</tr>
<tr>
<td>start_form_tag</td>
<td>12</td>
</tr>
<tr>
<td>strftime</td>
<td>16</td>
</tr>
<tr>
<td>stylesheet_link_tag</td>
<td>11</td>
</tr>
<tr>
<td>submit_tag</td>
<td>12</td>
</tr>
<tr>
<td>T</td>
<td>11</td>
</tr>
<tr>
<td>Template</td>
<td>11</td>
</tr>
<tr>
<td>text_field</td>
<td>13</td>
</tr>
<tr>
<td>U</td>
<td>38</td>
</tr>
<tr>
<td>update_attribute</td>
<td>38</td>
</tr>
<tr>
<td>update_attributes</td>
<td>10</td>
</tr>
<tr>
<td>updated_at</td>
<td>5</td>
</tr>
<tr>
<td>updated_on</td>
<td>5, 13</td>
</tr>
<tr>
<td>url_for</td>
<td>25</td>
</tr>
<tr>
<td>V</td>
<td>20</td>
</tr>
<tr>
<td>validates_associated</td>
<td>20</td>
</tr>
<tr>
<td>validates_format_of</td>
<td>20</td>
</tr>
<tr>
<td>validates_inclusion_of</td>
<td>20</td>
</tr>
<tr>
<td>validates_length_of</td>
<td>6, 20</td>
</tr>
<tr>
<td>validates_presence_of</td>
<td>20</td>
</tr>
<tr>
<td>validates_uniqueness_of</td>
<td>6</td>
</tr>
<tr>
<td>.each</td>
<td>38</td>
</tr>
</tbody>
</table>