

Keeping Your Views DRY*

Building Reusable Web Interfaces Using Views, Partial, Layouts, and Helpers

East Bay Ruby Meetup

November 21, 2006

*"DRY" = "Don't Repeat Yourself"

Introduction

- Motivated by a practical problem faced by all (Rails) programmers: how to build reusable views (in Rails).
- Based on our first-time experience doing this in Rails.

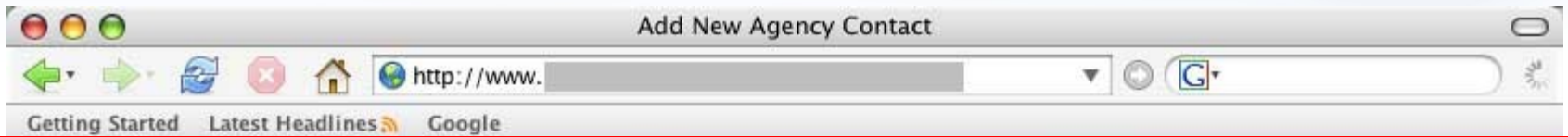
Introduction

- This presentation assumes basic knowledge about Rails (models, views, controllers, how to generate scaffolding).
- Attempt to bring the level down to “what's obvious” to experienced Rails developers.
- Challenge – to “make sense of all the pieces” - LOTS of details!

Background

- Web pages always contain elements that appear repeatedly on different pages.
- Examples:
 - Page titles
 - Page headings: graphics, text, toolbars,
 - Navigation elements: breadcrumbs, menu items, links.
 - Page footers: links, copyright notices.

Example



Header

November 20, 2006

[Dashboard](#) > [List Agency Contacts](#) > Add New Agency Contact

Breadcrumb

Add New Agency Contact

Title

Toolbar



Fill in the following form and click the Create button to add a new Agency Contact.

Agency contact firstname <input type="text"/>	Agency contact lastname <input type="text"/>	Agency Location SSLA <input type="text"/>
--	---	--

Body

[Cancel and Return to Agency Contacts list](#)

©Client

Footer

[Website](#) [Dashboard](#) [Sign Off](#)

Done

Problem

- Example: changing a link that appears on every page.
- “Repeating yourself” has two major disadvantages:
 - You have to copy and paste every element you need to replicate, every time you need to replicate it.
 - Changes to individual elements need to be made at every place they appear.
 - You are “wet.”

Typical “Drier” Solution

- Use templates and include files (e.g., PHP)
 - Place the common code in a separate include file.
 - Place a statement or directive in a template page that “includes the include file” wherever necessary
- Example: place page header in a separate file, include in every page that requires it. You don't have to repeat the page header code in every page.

DRY Views in Rails

- Rails provides a set of mechanisms to do the above.
- It is much more powerful, and fits nicely with Rails' Model View Controller (MVC) architecture:

Views, Partials, Layouts, and Helpers

Problem

new.rhtml

list.rhtml

_form.rhtml

Views

Partials

styles

Layouts

Helpers

edit.rhtml

Models

show.rhtml

controllers



Example: Scaffolding

- When scaffolding is generated, views to Create, Read, Update, and Delete (CRUD) files are created on the view directory:
 - list.rhtml
 - new.rhtml
 - edit.rhtml
 - show.rhtml
 - _form.rhtml

Start with a Layout

- A layout is basically a *master template page*, on which you include view elements when you create DRY views.
- The `app/views/layouts` directory is created when your rails application is first generated, but is empty.
- If you generate scaffolding for a type, e.g., “currency,” then a layout `app/views/layouts/currency.rhtml` is created (more on this later).

```
<html>
```

```
<head>
```

```
  <title>Currency: <%= controller.action_name %></title>
```

```
</head>
```

```
<body>
```

```
  <%= @content_for_layout %>
```

```
    <%= yield %>
```

```
</body>
```

```
</html>
```

Layout Content

- The content for the layout (such as list, edit, new, show) is expanded into the layout at the place where `@content_for_layout` or `yield` is called.
- In other words, the content in `list.rhtml`, `edit.rhtml`, `new.rhtml`, or `show.rhtml` are inserted into the layout when the controller action (list, edit, new, show) is executed.

Content for Layout Rendered

list.rhtml



currency/list.rhtml

@content_for_layout

Using Standard Layout

- You can create a layout named `standard.rhtml` in the `app/views/layouts` directory.
- In every *controller* that you want to use the standard layout “template,” include the line:

```
class CurrencyController < ApplicationController  
  layout 'standard'  
end
```

Default vs. Non-default Layouts

- If you do not supply an explicit layout, then the default view-specific layout will be automatically used to render the page.
- If you do specify a layout using a call to the layout method in the controller, then that layout will be used in place of the default layout.

Partials – Rails “include files”

- View-specific *partials* are created in a specific view (e.g., currency) directory. They are named using an “underscore character” (such as `_form.rhtml`) and are rendered using

```
<%= render :partial => 'form' %>
```

Shared Partials

- Shared partials are created in the `app/views/shared` directory.
- They can be used in any view:

```
<%= render :partial => 'shared/header' %>
```

Partials Rendered in Layout

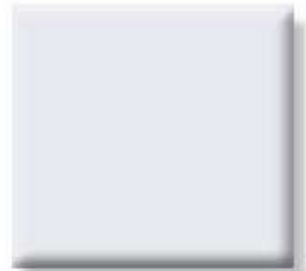
list.rhtml



currency/list.rhtml

```
<%= render :partial  
=> 'form' %>  
@content_for_layout
```

_form.rhtml



Helpers

- Helpers are located in `app/helpers`
- They are designed to help display views, but *minimize code in views*
- Helper code and variables are visible in the respective view (`currency_helper.rb`).

```
module CurrencyHelper  
end
```

- The application-wide helper is named `application_helper.rb` – its code and variables are available in every view.

Putting It All Together

- Example applications
- Page header graphics
- Page title
- Toolbar
- “Breadcrumb” navigation
[Dashboard](#) > [List Clients](#) > Edit Client
- Footer

Thank You

- A copy of this presentation will be posted at:

<http://www.dsdinteractive.com/dryviews.pdf>