**Rails Application Layout**

**by Daniel Kehoe**

*Last updated 8 December 2013*

Rails application layout for HTML5. Shows how to set up an application layout with navigation links, messages for alerts and notices, and CSS styling for Rails 4.0.

**If You Are New to Rails**

If you’re new to Rails, see [What is Ruby on Rails?](http://railsapps.github.io/what-is-ruby-rails.html), the book [Learn Ruby on Rails](http://learn-rails.com/learn-ruby-on-rails.html), and recommendations for a [Rails tutorial](https://tutorials.railsapps.org/rails-tutorial).

**[](http://learn-rails.com/learn-ruby-on-rails.html)**

**What is the RailsApps Project?**

This is an article from the RailsApps project. The [RailsApps project](http://railsapps.github.io/) provides example applications that developers use as starter apps. Hundreds of developers use the apps, report problems as they arise, and propose solutions. Rails changes frequently; each application is known to work and serves as your personal “reference implementation” so you can stay up to date. Each is accompanied by a tutorial so there is no mystery code. Support for the project, including the example applications and the [Rails Composer](http://railsapps.github.io/rails-composer/) tool, comes from subscribers to the [RailsApps tutorials](https://tutorials.railsapps.org).

**Background**

The default application layout is where you put HTML that you want to include on every page of your website.

Every Rails application needs a well-designed application layout. The Rails default starter application, which is created when you run rails new, provides a barebones application layout. You’ll want to add navigation links, include flash messages for errors and notifications, and apply CSS styling.

Rails will use the layout defined in the file **app/views/layouts/application.html.erb** as a default for rendering any page. See the RailsGuide [Layouts and Rendering in Rails](http://guides.rubyonrails.org/layouts_and_rendering.html#using-render) for details. The book [Learn Ruby on Rails](http://learn-rails.com/learn-ruby-on-rails.html) provides an in-depth explanation.

**HTML5 Boilerplate**

The well-known HTML5 Boilerplate project has been recommending “best practice” tweaks to web pages since 2010. Very few of the HTML5 Boilerplate recommendations are relevant for Rails developers, as Rails already provides almost everything required. If you want to learn more, the article [HTML5 Boilerplate for Rails Developers](http://railsapps.github.io/rails-html5-boilerplate.html) looks at the recommendations.

**Front-end Frameworks**

This article shows how to set up a simple application layout with navigation links, messages for alerts and notices, and CSS styling for Rails 4.0. If you intend to use a front-end framework such as Twitter Bootstrap or Zurb Foundation, see these articles for information about how to set up the application layout:

* [**Rails and Bootstrap**](http://railsapps.github.io/twitter-bootstrap-rails.html)
* [**Rails and Foundation**](http://railsapps.github.io/rails-foundation.html)

**Rails Layout Gem**

This article shows how to use the [rails\_layout](https://github.com/RailsApps/rails_layout) gem to generate files for an application layout, navigation links, and flash messages styled with simple CSS classes and layout. You don’t have to use the gem; you can copy the code directly from this article.

**Rails Default Application Layout**

Generating a new Rails application with the rails new command will create an application layout in the file **app/views/layouts/application.html.erb**. Here is the default with Rails 4.0:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Myapp</title>  <%= stylesheet\_link\_tag "application", media: "all", "data-turbolinks-track" => true %>  <%= javascript\_include\_tag "application", "data-turbolinks-track" => true %>  <%= csrf\_meta\_tags %>  </head>  <body>  <%= yield %>  </body>  </html> |

This is a barebones application layout.

It contains a title for your web pages.

The stylesheet\_link\_tag and javascript\_include\_tag elements add CSS stylesheets and JavaScript from the [Rails asset pipeline](http://guides.rubyonrails.org/asset_pipeline.html). The data-turbolinks-track attribute supports Rails [Turbolinks](https://github.com/rails/turbolinks/blob/master/README.md).

The csrf\_meta\_tags element is a view helper that prevents [cross-site request forgeries](https://en.wikipedia.org/wiki/Cross-site_request_forgery) when users input data in forms.

The content from a Rails view is inserted where you place the yield keyword. See the RailsGuide [Layouts and Rendering in Rails](http://guides.rubyonrails.org/layouts_and_rendering.html#using-render) for details about how Rails combines views with layouts.

**The Rails Layout Gem**

You can use the [rails\_layout](https://github.com/RailsApps/rails_layout) gem to generate files for an improved application layout. It will add navigation links and flash messages plus simple CSS classes for styling.

In your **Gemfile**, add:

|  |
| --- |
| group :development do  gem 'rails\_layout'  end |

You don’t need the [rails\_layout](https://github.com/RailsApps/rails_layout) gem deployed to production, so put it in the development group.

Run

|  |
| --- |
| $ bundle install |

in the Terminal.

The rails\_layout gem uses the rails generate command to create files. Run:

|  |
| --- |
| $ rails generate layout simple --force |

The --force argument will force the gem to replace the existing **app/views/layouts/application.html.erb** file.

The gem will add five files to your project:

* **app/views/layouts/application.html.erb**
* **app/views/layouts/\_messages.html.erb**
* **app/views/layouts/\_navigation.html.erb**
* **app/views/layouts/\_navigation\_links.html.erb**
* **app/assets/stylesheets/simple.css**

Next we’ll examine the files created by the [rails\_layout](https://github.com/RailsApps/rails_layout) gem.

**Improved Application Layout**

Here is the file **app/views/layouts/application.html.erb** created by the [rails\_layout](https://github.com/RailsApps/rails_layout) gem:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title><%= content\_for?(:title) ? yield(:title) : "My Application" %></title>  <meta name="description" content="<%= content\_for?(:description) ? yield(:description) : "Learn Rails" %>">  <%= stylesheet\_link\_tag "application", media: "all", "data-turbolinks-track" => true %>  <%= javascript\_include\_tag "application", "data-turbolinks-track" => true %>  <%= csrf\_meta\_tags %>  </head>  <body>  <header>  <%= render 'layouts/navigation' %>  </header>  <main role="main">  <%= render 'layouts/messages' %>  <%= yield %>  </main>  </body>  </html> |

**Viewport**

The viewport metatag improves the presentation of web pages on mobile devices. Setting a viewport tells the browser how content should fit on the device’s screen. Apple’s developer documentation on [Configuring the Viewport](http://developer.apple.com/library/safari/#documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html) provides details.

**Title and Description**

The *title* and *description* metatags help make your web pages search-engine friendly. Google uses contents of the title tag to display titles in search results. And it will sometimes use the content of a description metatag in search results snippets. See Google’s explanation of how it uses [Site Title and Description](http://support.google.com/webmasters/bin/answer.py?hl=en&answer=35624).

The code uses the Ruby [ternary operator](http://ruby.about.com/od/beginningruby/a/The-Ternary-Or-Conditional-Operator.htm). It is a conditional statement that says, “if content\_for?(:title) is present in the view file, use yield(:title) to include it, otherwise just display ‘My Application’.”

You can use a content\_for statement to set a title and description on each individual page.

The combination of content\_for? with yield allows you to inject data into the application layout from any view. Set up a view file containing the code:

|  |
| --- |
| <% content\_for :title do %>My Title<% end %>  <% content\_for :description do %>My Description<% end %> |

When a controller renders the view template and combines it with the application layout, the text “My Title” will replace <%= content\_for?(:title) ? yield(:title) : "My Application" %> in the application layout. If no content\_for block is found in the view template, the alternate text “My Application” will be used.

**Body Tag**

The simplest form of the <body> tag is all you need for many applications. However, if you’re using JavaScript extensively and wish to execute JavaScript conditionally on a page, it’s helpful to set the controller name and action in the <body> tag:

|  |
| --- |
| <body class="<%= controller\_name %> <%= action\_name %>"> |

See the article [Adding JavaScript to Rails](http://railsapps.github.io/rails-javascript-include-external.html) for details.

**HTML5 Elements**

The improved application layout contains tags that are structural elements from the HTML5 specification:

* <header>
* <main>

These elements add structure to a web page. The tags don’t add any new behavior but make it easier to determine the structure of the page and apply CSS styles.

The <header> tag is typically used for branding or navigation.

We wrap a navigation partial in the <header> tag:

|  |
| --- |
| <header>  <%= render 'layouts/navigation' %>  </header> |

The <main> tag is among the newest HTML5 elements (see the [W3C specification](http://www.w3.org/html/wg/drafts/html/master/grouping-content.html#the-main-element) for details). From the specification: “The main content area of a document includes content that is unique to that document and excludes content that is repeated across a set of documents such as site navigation links, copyright information, site logos.” We follow the advice of the specification and wrap our unique content in the <main> tag.

The specification recommends, “Authors are advised to use ARIA role=‘main’ attribute on the main element until user agents implement the required role mapping.” [ARIA](http://www.w3.org/WAI/intro/aria), the Accessible Rich Internet Applications Suite, is a specification to make web applications more accessible to people with disabilities. That means the role="main" attribute is there for any web browsers that don’t yet recognize the <main> tag, and may help people with disabilities.

|  |
| --- |
| <main role="main">  <%= render 'layouts/messages' %>  <%= yield %>  </main> |

We wrap a messages partial and yield expression in a <main role="main"> element.

You could add a <footer> tag. It typically contains links to copyright information, legal disclaimers, or contact information.

**Navigation Partial**

You’ll likely need navigation links on every page of your web application.

You can add navigation links directly to your application layout file but many developers prefer to create a [partial template](http://guides.rubyonrails.org/layouts_and_rendering.html#using-partials) – a “partial” – to better organize the default application layout.

Examine the file **app/views/layouts/\_navigation.html.erb**:

|  |
| --- |
| <ul class="nav">  <li><%= link\_to 'Home', root\_path %></li>  <%= render 'layouts/navigation\_links' %>  </ul> |

The navigation partial contains a link\_to helper for a navigation link to the application home page. You’ll need to set the route root\_path in the **config/routes.rb** file.

The navigation partial includes a nested partial, the navigation links partial, in the file **app/views/layouts/\_navigation\_links.html.erb**:

|  |
| --- |
| <%# add navigation links to this file %> |

As you add pages to your application, you can add links to this file.

In a simple application, it is not necessary to nest a navigation links partial. Use it to isolate the complex markup required by a front-end framework from a simple list of links needed for navigation, if you wish.

**Flash Messages**

Rails provides a standard convention to display alerts (including error messages) and other notices (including success messages), called a *flash message*. The name comes from the term “flash memory” and should not be confused with the “Adobe Flash” web development platform that was once popular for animated websites. The flash message is documented in the [RailsGuides: Action Controller Overview](http://guides.rubyonrails.org/action_controller_overview.html#the-flash).

Code to display flash messages can go directly in your application layout file or you can use a partial.

Here the application layout references a partial contained in the file **app/views/layouts/\_messages.html.erb**:

|  |
| --- |
| <% flash.each do |name, msg| %>  <% if msg.is\_a?(String) %>  <%= content\_tag :div, msg, :class => "flash\_#{name}" %>  <% end %>  <% end %> |

We use each to iterate through the flash hash, retrieving a name and msg that are passed to a block to be output as a string. The expression if msg.is\_a?(String) serves as a test to make sure we only display messages that are strings. We use the Rails content\_tag view helper to create the HTML div. Finally, we apply a CSS class and combine the word “flash” with “notice” or “alert” to make the CSS class.

**Simple CSS**

The [rails\_layout](https://github.com/RailsApps/rails_layout) gem adds a CSS file for simple styling of the navigation links and flash messages. If you don’t want to use Twitter Bootstrap, Zurb Foundation, or another CSS front-end framework, you can apply simple CSS styling for your navigation links and flash messages with the following code.

Examine the file **app/assets/stylesheets/simple.css**:

|  |
| --- |
| /\*  \* Simple CSS stylesheet for a navigation bar and flash messages.  \*/  main {  background-color: #eee;  padding-bottom: 80px;  width: 100%;  }  header {  border: 1px solid #d4d4d4;  background-image: linear-gradient(to bottom, white, #f2f2f2);  background-color: #f9f9f9;  -webkit-box-shadow: 0 1px 10px rgba(0, 0, 0, 0.1);  -moz-box-shadow: 0 1px 10px rgba(0, 0, 0, 0.1);  box-shadow: 0 1px 10px rgba(0, 0, 0, 0.1);  margin-bottom: 20px;  font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;  }  ul.nav li {  display: inline;  }  ul.nav li a {  padding: 10px 15px 10px;  color: #777777;  text-decoration: none;  text-shadow: 0 1px 0 white;  }  .flash\_notice, .flash\_alert {  padding: 8px 35px 8px 14px;  margin-bottom: 20px;  text-shadow: 0 1px 0 rgba(255, 255, 255, 0.5);  border: 1px solid #fbeed5;  -webkit-border-radius: 4px;  -moz-border-radius: 4px;  border-radius: 4px;  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;  font-size: 14px;  line-height: 20px;  }  .flash\_notice {  background-color: #dff0d8;  border-color: #d6e9c6;  color: #468847;  }  .flash\_alert {  background-color: #f2dede;  border-color: #eed3d7;  color: #b94a48;  } |

The CSS rules set a background color for the main section of the page, plus styles for a header, navigation links, and flash messages.

**Did You Like the Article?**

Was the article useful to you? Follow [@rails\_apps](http://twitter.com/rails_apps) on Twitter and tweet some praise. I’d love to know you were helped out by the article.

You can also find me on [Facebook](https://www.facebook.com/daniel.kehoe.sf) or [Google+](https://plus.google.com/+DanielKehoe/).