Action View Overview

January 13, 2015

After reading this guide, you will know:

- What Action View is and how to use it with Rails.
- How best to use templates, partials, and layouts.
- What helpers are provided by Action View and how to make your own.
- How to use localized views.

1 What is Action View?

Action View and Action Controller are the two major components of Action Pack. In Rails, web requests are handled by Action Pack, which splits the work into a controller part (performing the logic) and a view part (rendering a template). Typically, Action Controller will be concerned with communicating with the database and performing CRUD actions where necessary. Action View is then responsible for compiling the response.

Action View templates are written using embedded Ruby in tags mingled with HTML. To avoid cluttering the templates with boilerplate code, a number of helper classes provide common behavior for forms, dates, and strings. It's also easy to add new helpers to your application as it evolves.

Some features of Action View are tied to Active Record, but that doesn't mean Action View depends on Active Record. Action View is an independent package that can be used with any sort of Ruby libraries.

2 Using Action View with Rails

For each controller there is an associated directory in the `app/views` directory which holds the template files that make up the views associated with that controller. These files are used to display the view that results from each controller action.

Let's take a look at what Rails does by default when creating a new resource using the scaffold generator:

```
$ bin/rails generate scaffold article
      [...]
      invoke  scaffold_controller
      create    app/controllers/articles_controller.rb
      invoke    erb
      create      app/views/articles
      create      app/views/articles/index.html.erb
      create      app/views/articles/edit.html.erb
```

```
create      app/views/articles/show.html.erb
create      app/views/articles/new.html.erb
create      app/views/articles/_form.html.erb
[...]
```

There is a naming convention for views in Rails. Typically, the views share their name with the associated controller action, as you can see above. For example, the index controller action of the `articles_controller.rb` will use the `index.html.erb` view file in the `app/views/articles` directory. The complete HTML returned to the client is composed of a combination of this ERB file, a layout template that wraps it, and all the partials that the view may reference. Within this guide you will find more detailed documentation about each of these three components.

3 Templates, Partials and Layouts

As mentioned, the final HTML output is a composition of three Rails elements: `Templates`, `Partials` and `Layouts`. Below is a brief overview of each of them.

3.1 Templates

Action View templates can be written in several ways. If the template file has a `.erb` extension then it uses a mixture of ERB (Embedded Ruby) and HTML. If the template file has a `.builder` extension then the `Builder::XmlMarkup` library is used.

Rails supports multiple template systems and uses a file extension to distinguish amongst them. For example, an HTML file using the ERB template system will have `.html.erb` as a file extension.

3.1.1 ERB Within an ERB template, Ruby code can be included using both <% %> and <%= %> tags. The <% %> tags are used to execute Ruby code that does not return anything, such as conditions, loops or blocks, and the <%= %> tags are used when you want output.

Consider the following loop for names:

```
<h1>Names of all the people</h1>
<% @people.each do |person| %>
  Name: <%= person.name %><br>
<% end %>
```

The loop is set up using regular embedding tags (<% %>) and the name is inserted using the output embedding tags (<%= %>). Note that this is not just a usage suggestion: regular output functions such as `print` and `puts` won't be rendered to the view with ERB templates. So this would be wrong:

```
<%# WRONG %>
Hi, Mr. <% puts "Frodo" %>
```

To suppress leading and trailing whitespaces, you can use <%- -%> interchangeably with <% and %>.

3.1.2 Builder Builder templates are a more programmatic alternative to ERB. They are especially useful for generating XML content. An XmlMarkup object named `xml` is automatically made available to templates with a `.builder` extension.

Here are some basic examples:

```
xml.em("emphasized")
xml.em { xml.b("emph & bold") }
xml.a("A Link", "href" => "http://rubyonrails.org")
xml.target("name" => "compile", "option" => "fast")
```

which would produce:

```
<em>emphasized</em>
<em><b>emph &amp; bold</b></em>
<a href="http://rubyonrails.org">A link</a>
<target option="fast" name="compile" />
```

Any method with a block will be treated as an XML markup tag with nested markup in the block. For example, the following:

```
xml.div {
  xml.h1(@person.name)
  xml.p(@person.bio)
}
```

would produce something like:

```
<div>
  <h1>David Heinemeier Hansson</h1>
  <p>A product of Danish Design during the Winter of '79...</p>
</div>
```

Below is a full-length RSS example actually used on Basecamp:

```
xml.rss("version" => "2.0", "xmlns:dc" => "http://purl.org/dc/elements/1.1/") do
  xml.channel do
    xml.title(@feed_title)
    xml.link(@url)
    xml.description "Basecamp: Recent items"
    xml.language "en-us"
    xml.ttl "40"

    for item in @recent_items
      xml.item do
        xml.title(item_title(item))
        xml.description(item_description(item)) if item_description(item)
        xml.pubDate(item_pubDate(item))
        xml.guid(@person.firm.account.url + @recent_items.url(item))
```

```
        xml.link(@person.firm.account.url + @recent_items.url(item))
        xml.tag!("dc:creator", item.author_name) if item_has_creator?(item)
      end
    end
  end
end
```

3.1.3 Template Caching By default, Rails will compile each template to a method in order to render it. When you alter a template, Rails will check the file's modification time and recompile it in development mode.

3.2 Partials

Partial templates - usually just called "partials" - are another device for breaking the rendering process into more manageable chunks. With partials, you can extract pieces of code from your templates to separate files and also reuse them throughout your templates.

3.2.1 Naming Partials To render a partial as part of a view, you use the `render` method within the view:

```
<%= render "menu" %>
```

This will render a file named `_menu.html.erb` at that point within the view that is being rendered. Note the leading underscore character: partials are named with a leading underscore to distinguish them from regular views, even though they are referred to without the underscore. This holds true even when you're pulling in a partial from another folder:

```
<%= render "shared/menu" %>
```

That code will pull in the partial from `app/views/shared/_menu.html.erb`.

3.2.2 Using Partials to simplify Views One way to use partials is to treat them as the equivalent of subroutines; a way to move details out of a view so that you can grasp what's going on more easily. For example, you might have a view that looks like this:

```
<%= render "shared/ad_banner" %>

<h1>Products</h1>

<p>Here are a few of our fine products:</p>
<% @products.each do |product| %>
  <%= render partial: "product", locals: {product: product} %>
<% end %>

<%= render "shared/footer" %>
```

Here, the `_ad_banner.html.erb` and `_footer.html.erb` partials could contain content that is shared among many pages in your application. You don't need to see the details of these sections when you're concentrating on a particular page.

3.2.3 The `as` and `object` options By default `ActionView::Partials::PartialRenderer` has its object in a local variable with the same name as the template. So, given:

```
<%= render partial: "product" %>
```

within product we'll get `@product` in the local variable `product`, as if we had written:

```
<%= render partial: "product", locals: {product: @product} %>
```

With the `as` option we can specify a different name for the local variable. For example, if we wanted it to be `item` instead of `product` we would do:

```
<%= render partial: "product", as: "item" %>
```

The `object` option can be used to directly specify which object is rendered into the partial; useful when the template's object is elsewhere (eg. in a different instance variable or in a local variable).

For example, instead of:

```
<%= render partial: "product", locals: {product: @item} %>
```

we would do:

```
<%= render partial: "product", object: @item %>
```

The `object` and `as` options can also be used together:

```
<%= render partial: "product", object: @item, as: "item" %>
```

3.2.4 Rendering Collections It is very common that a template will need to iterate over a collection and render a sub-template for each of the elements. This pattern has been implemented as a single method that accepts an array and renders a partial for each one of the elements in the array.

So this example for rendering all the products:

```
<% @products.each do |product| %>
  <%= render partial: "product", locals: { product: product } %>
<% end %>
```

can be rewritten in a single line:

```
<%= render partial: "product", collection: @products %>
```

When a partial is called with a collection, the individual instances of the partial have access to the member of the collection being rendered via a variable named after the partial. In this case, the partial is `_product`, and within it you can refer to `product` to get the collection member that is being rendered.

You can use a shorthand syntax for rendering collections. Assuming `@products` is a collection of `Product` instances, you can simply write the following to produce the same result:

```
<%= render @products %>
```

Rails determines the name of the partial to use by looking at the model name in the collection, `Product` in this case. In fact, you can even render a collection made up of instances of different models using this shorthand, and Rails will choose the proper partial for each member of the collection.

3.2.5 Spacer Templates You can also specify a second partial to be rendered between instances of the main partial by using the `:spacer_template` option:

```
<%= render partial: @products, spacer_template: "product_ruler" %>
```

Rails will render the `_product_ruler` partial (with no data passed to it) between each pair of `_product` partials.

3.3 Layouts

Layouts can be used to render a common view template around the results of Rails controller actions. Typically, a Rails application will have a couple of layouts that pages will be rendered within. For example, a site might have one layout for a logged in user and another for the marketing or sales side of the site. The logged in user layout might include top-level navigation that should be present across many controller actions. The sales layout for a SaaS app might include top-level navigation for things like "Pricing" and "Contact Us" pages. You would expect each layout to have a different look and feel. You can read about layouts in more detail in the Layouts and Rendering in Rails guide.

4 Partial Layouts

Partials can have their own layouts applied to them. These layouts are different from those applied to a controller action, but they work in a similar fashion.

Let's say we're displaying an article on a page which should be wrapped in a `div` for display purposes. Firstly, we'll create a new `Article`:

```
Article.create(body: 'Partial Layouts are cool!')
```

In the `show` template, we'll render the `_article` partial wrapped in the `box` layout:
articles/show.html.erb

```
<%= render partial: 'article', layout: 'box', locals: {article: @article} %>
```

The `box` layout simply wraps the `_article` partial in a `div`:
articles/_box.html.erb

```
<div class='box'>
  <%= yield %>
</div>
```

The `_article` partial wraps the article's `body` in a `div` with the `id` of the article using the `div_for` helper:
articles/_article.html.erb

```
<%= div_for(article) do %>
  <p><%= article.body %></p>
<% end %>
```

this would output the following:

```
<div class='box'>
  <div id='article_1'>
    <p>Partial Layouts are cool!</p>
  </div>
</div>
```

Note that the partial layout has access to the local `article` variable that was passed into the `render` call. However, unlike application-wide layouts, partial layouts still have the underscore prefix.

You can also render a block of code within a partial layout instead of calling `yield`. For example, if we didn't have the `_article` partial, we could do this instead:

articles/show.html.erb

```
<% render(layout: 'box', locals: {article: @article}) do %>
  <%= div_for(article) do %>
    <p><%= article.body %></p>
  <% end %>
<% end %>
```

Supposing we use the same `_box` partial from above, this would produce the same output as the previous example.

5 View Paths

..

6 Overview of helpers provided by Action View

WIP: Not all the helpers are listed here. For a full list see the API documentation

The following is only a brief overview summary of the helpers available in Action View. It's recommended that you review the API Documentation, which covers all of the helpers in more detail, but this should serve as a good starting point.

6.1 RecordTagHelper

This module provides methods for generating container tags, such as `div`, for your record. This is the recommended way of creating a container for render your Active Record object, as it adds an appropriate class and id attributes to that container. You can then refer to those containers easily by following the convention, instead of having to think about which class or id attribute you should use.

6.1.1 content_tag_for Renders a container tag that relates to your Active Record Object.

For example, given `@article` is the object of `Article` class, you can do:

```
<%= content_tag_for(:tr, @article) do %>
  <td><%= @article.title %></td>
<% end %>
```

This will generate this HTML output:

```
<tr id="article_1234" class="article">
  <td>Hello World!</td>
</tr>
```

You can also supply HTML attributes as an additional option hash. For example:

```
<%= content_tag_for(:tr, @article, class: "frontpage") do %>
  <td><%= @article.title %></td>
<% end %>
```

Will generate this HTML output:

```
<tr id="article_1234" class="article frontpage">
  <td>Hello World!</td>
</tr>
```

You can pass a collection of Active Record objects. This method will loop through your objects and create a container for each of them. For example, given `@articles` is an array of two `Article` objects:

```
<%= content_tag_for(:tr, @articles) do |article| %>
  <td><%= article.title %></td>
<% end %>
```

Will generate this HTML output:

```
<tr id="article_1234" class="article">
  <td>Hello World!</td>
</tr>
<tr id="article_1235" class="article">
  <td>Ruby on Rails Rocks!</td>
</tr>
```

6.1.2 div_for This is actually a convenient method which calls `content_tag_for` internally with `:div` as the tag name. You can pass either an Active Record object or a collection of objects. For example:

```
<%= div_for(@article, class: "frontpage") do %>
  <td><%= @article.title %></td>
<% end %>
```

Will generate this HTML output:

```
<div id="article_1234" class="article frontpage">
  <td>Hello World!</td>
</div>
```

6.2 AssetTagHelper

This module provides methods for generating HTML that links views to assets such as images, JavaScript files, stylesheets, and feeds.

By default, Rails links to these assets on the current host in the public folder, but you can direct Rails to link to assets from a dedicated assets server by setting `config.action_controller.asset_host` in the application configuration, typically in `config/environments/production.rb`. For example, let's say your asset host is `assets.example.com`:

```
config.action_controller.asset_host = "assets.example.com"
image_tag("rails.png") # => <img src="http://assets.example.com/images/rails.png"
alt="Rails" />
```

6.2.1 register_javascript_expansion Register one or more JavaScript files to be included when symbol is passed to javascript_include_tag. This method is typically intended to be called from plugin initialization to register JavaScript files that the plugin installed in `vendor/assets/javascripts`.

```
ActionView::Helpers::AssetTagHelper.register_javascript_expansion monkey: ["head", "body",
"tail"]

javascript_include_tag :monkey # =>
  <script src="/assets/head.js"></script>
  <script src="/assets/body.js"></script>
  <script src="/assets/tail.js"></script>
```

6.2.2 register_stylesheet_expansion Register one or more stylesheet files to be included when symbol is passed to `stylesheet_link_tag`. This method is typically intended to be called from plugin initialization to register stylesheet files that the plugin installed in `vendor/assets/stylesheets`.

```
ActionView::Helpers::AssetTagHelper.register_stylesheet_expansion monkey: ["head", "body",
"tail"]

stylesheet_link_tag :monkey # =>
  <link href="/assets/head.css" media="screen" rel="stylesheet" />
  <link href="/assets/body.css" media="screen" rel="stylesheet" />
  <link href="/assets/tail.css" media="screen" rel="stylesheet" />
```

6.2.3 auto_discovery_link_tag Returns a link tag that browsers and feed readers can use to auto-detect an RSS or Atom feed.

```
auto_discovery_link_tag(:rss, "http://www.example.com/feed.rss", {title: "RSS Feed"}) # =
>
  <link rel="alternate" type="application/rss+xml" title="RSS Feed" href="http://
www.example.com/feed" />
```

6.2.4 image_path Computes the path to an image asset in the `app/assets/images` directory. Full paths from the document root will be passed through. Used internally by `image_tag` to build the image path.

```
image_path("edit.png") # => /assets/edit.png
```

Fingerprint will be added to the filename if config.assets.digest is set to true.

```
image_path("edit.png") # => /assets/edit-2d1a2db63fc738690021fedb5a65b68e.png
```

6.2.5 image_url Computes the url to an image asset in the `app/assets/images` directory. This will call `image_path` internally and merge with your current host or your asset host.

```
image_url("edit.png") # => http://www.example.com/assets/edit.png
```

6.2.6 image_tag Returns an HTML image tag for the source. The source can be a full path or a file that exists in your `app/assets/images` directory.

```
image_tag("icon.png") # => <img src="/assets/icon.png" alt="Icon" />
```

6.2.7 javascript_include_tag Returns an HTML script tag for each of the sources provided. You can pass in the filename (`.js` extension is optional) of JavaScript files that exist in your `app/assets/javascripts` directory for inclusion into the current page or you can pass the full path relative to your document root.

```
javascript_include_tag "common" # => <script src="/assets/common.js"></
script>
```

If the application does not use the asset pipeline, to include the jQuery JavaScript library in your application, pass `:defaults` as the source. When using `:defaults`, if an `application.js` file exists in your `app/assets/javascripts` directory, it will be included as well.

```
javascript_include_tag :defaults
```

You can also include all JavaScript files in the `app/assets/javascripts` directory using `:all` as the source.

```
javascript_include_tag :all
```

You can also cache multiple JavaScript files into one file, which requires less HTTP connections to download and can better be compressed by gzip (leading to faster transfers). Caching will only happen if `ActionController::Base.perform_caching` is set to true (which is the case by default for the Rails production environment, but not for the development environment).

```
javascript_include_tag :all, cache: true # =>
  <script src="/javascripts/all.js"></script>
```

6.2.8 javascript_path Computes the path to a JavaScript asset in the `app/assets/javascripts` directory. If the source filename has no extension, `.js` will be appended. Full paths from the document root will be passed through. Used internally by `javascript_include_tag` to build the script path.

```
javascript_path "common" # => /assets/common.js
```

6.2.9 javascript_url Computes the url to a JavaScript asset in the `app/assets/javascripts` directory. This will call `javascript_path` internally and merge with your current host or your asset host.

```
javascript_url "common" # => http://www.example.com/assets/common.js
```

6.2.10 stylesheet_link_tag Returns a stylesheet link tag for the sources specified as arguments. If you don't specify an extension, `.css` will be appended automatically.

```
stylesheet_link_tag "application" # => <link href="/assets/application.css" media=
"screen" rel="stylesheet" />
```

You can also include all styles in the stylesheet directory using :all as the source:

```
stylesheet_link_tag :all
```

You can also cache multiple stylesheets into one file, which requires less HTTP connections and can better be compressed by gzip (leading to faster transfers). Caching will only happen if ActionController::Base.perform_caching is set to true (which is the case by default for the Rails production environment, but not for the development environment).

```
stylesheet_link_tag :all, cache: true
# => <link href="/assets/all.css" media="screen" rel="stylesheet" />
```

6.2.11 stylesheet_path Computes the path to a stylesheet asset in the `app/assets/stylesheets` directory. If the source filename has no extension, .css will be appended. Full paths from the document root will be passed through. Used internally by stylesheet_link_tag to build the stylesheet path.

```
stylesheet_path "application" # => /assets/application.css
```

6.2.12 stylesheet_url Computes the url to a stylesheet asset in the `app/assets/stylesheets` directory. This will call `stylesheet_path` internally and merge with your current host or your asset host.

```
stylesheet_url "application" # => http://www.example.com/assets/application.css
```

6.3 AtomFeedHelper

6.3.1 atom_feed This helper makes building an Atom feed easy. Here's a full usage example:
config/routes.rb

```
resources :articles
```

app/controllers/articles_controller.rb

```
def index
  @articles = Article.all

  respond_to do |format|
    format.html
    format.atom
  end
end
```

app/views/articles/index.atom.builder

```
atom_feed do |feed|
  feed.title("Articles Index")
  feed.updated((@articles.first.created_at))

  @articles.each do |article|
    feed.entry(article) do |entry|
      entry.title(article.title)
      entry.content(article.body, type: 'html')

      entry.author do |author|
        author.name(article.author_name)
      end
    end
  end
end
```

6.4 BenchmarkHelper

6.4.1 benchmark Allows you to measure the execution time of a block in a template and records the result to the log. Wrap this block around expensive operations or possible bottlenecks to get a time reading for the operation.

```
<% benchmark "Process data files" do %>
  <%= expensive_files_operation %>
<% end %>
```

This would add something like "Process data files (0.34523)" to the log, which you can then use to compare timings when optimizing your code.

6.5 CacheHelper

6.5.1 cache A method for caching fragments of a view rather than an entire action or page. This technique is useful caching pieces like menus, lists of news topics, static HTML fragments, and so on. This method takes a block that contains the content you wish to cache. See `ActionController::Caching::Fragments` for more information.

```
<% cache do %>
  <%= render "shared/footer" %>
<% end %>
```

6.6 CaptureHelper

6.6.1 capture The `capture` method allows you to extract part of a template into a variable. You can then use this variable anywhere in your templates or layout.

```
<% @greeting = capture do %>
  <p>Welcome! The date and time is <%= Time.now %></p>
<% end %>
```

The captured variable can then be used anywhere else.

```
<html>
  <head>
    <title>Welcome!</title>
  </head>
  <body>
    <%= @greeting %>
  </body>
</html>
```

6.6.2 content_for Calling `content_for` stores a block of markup in an identifier for later use. You can make subsequent calls to the stored content in other templates or the layout by passing the identifier as an argument to `yield`.

For example, let's say we have a standard application layout, but also a special page that requires certain JavaScript that the rest of the site doesn't need. We can use `content_for` to include this JavaScript on our special page without fattening up the rest of the site.

app/views/layouts/application.html.erb

```
<html>
  <head>
    <title>Welcome!</title>
    <%= yield :special_script %>
  </head>
  <body>
    <p>Welcome! The date and time is <%= Time.now %></p>
  </body>
</html>
```

app/views/articles/special.html.erb

```
<p>This is a special page.</p>

<% content_for :special_script do %>
  <script>alert('Hello!')</script>
<% end %>
```

6.7 DateHelper

6.7.1 date_select Returns a set of select tags (one for year, month, and day) pre-selected for accessing a specified date-based attribute.

```
date_select("article", "published_on")
```

6.7.2 datetime_select Returns a set of select tags (one for year, month, day, hour, and minute) pre-selected for accessing a specified datetime-based attribute.

```
datetime_select("article", "published_on")
```

6.7.3 distance_of_time_in_words Reports the approximate distance in time between two Time or Date objects or integers as seconds. Set `include_seconds` to true if you want more detailed approximations.

```
distance_of_time_in_words(Time.now, Time.now + 15.seconds)        # => less than a
minute
distance_of_time_in_words(Time.now, Time.now + 15.seconds, include_seconds: true)  # =>
less than 20 seconds
```

6.7.4 select_date Returns a set of HTML select-tags (one for year, month, and day) pre-selected with the `date` provided.

```
# Generates a date select that defaults to the date provided (six days after today)
select_date(Time.today + 6.days)

# Generates a date select that defaults to today (no specified date)
select_date()
```

6.7.5 select_datetime Returns a set of HTML select-tags (one for year, month, day, hour, and minute) pre-selected with the `datetime` provided.

```
# Generates a datetime select that defaults to the datetime provided (four days after
today)
select_datetime(Time.now + 4.days)

# Generates a datetime select that defaults to today (no specified datetime)
select_datetime()
```

6.7.6 select_day Returns a select tag with options for each of the days 1 through 31 with the current day selected.

```
# Generates a select field for days that defaults to the day for the date provided
select_day(Time.today + 2.days)

# Generates a select field for days that defaults to the number given
select_day(5)
```

6.7.7 select_hour Returns a select tag with options for each of the hours 0 through 23 with the current hour selected.

```
# Generates a select field for hours that defaults to the hours for the time provided
select_hour(Time.now + 6.hours)
```

6.7.8 select_minute Returns a select tag with options for each of the minutes 0 through 59 with the current minute selected.

```
# Generates a select field for minutes that defaults to the minutes for the time provided.
select_minute(Time.now + 6.hours)
```

6.7.9 select_month Returns a select tag with options for each of the months January through December with the current month selected.

```
# Generates a select field for months that defaults to the current month
select_month(Date.today)
```

6.7.10 select_second Returns a select tag with options for each of the seconds 0 through 59 with the current second selected.

```
# Generates a select field for seconds that defaults to the seconds for the time provided
select_second(Time.now + 16.minutes)
```

6.7.11 select_time Returns a set of HTML select-tags (one for hour and minute).

```
# Generates a time select that defaults to the time provided
select_time(Time.now)
```

6.7.12 select_year Returns a select tag with options for each of the five years on each side of the current, which is selected. The five year radius can be changed using the `:start_year` and `:end_year` keys in the `options`.

```
# Generates a select field for five years on either side of Date.today that defaults to
the current year
select_year(Date.today)

# Generates a select field from 1900 to 2009 that defaults to the current year
select_year(Date.today, start_year: 1900, end_year: 2009)
```

6.7.13 time_ago_in_words Like `distance_of_time_in_words`, but where `to_time` is fixed to `Time.now`.

```
time_ago_in_words(3.minutes.from_now)  # => 3 minutes
```

6.7.14 time_select Returns a set of select tags (one for hour, minute and optionally second) pre-selected for accessing a specified time-based attribute. The selects are prepared for multi-parameter assignment to an Active Record object.

```
# Creates a time select tag that, when POSTed, will be stored in the order variable in the
submitted attribute
time_select("order", "submitted")
```

6.8 DebugHelper

Returns a `pre` tag that has object dumped by YAML. This creates a very readable way to inspect an object.

```
my_hash = {'first' => 1, 'second' => 'two', 'third' => [1,2,3]}
debug(my_hash)
```

```
<pre class='debug_dump'>---
first: 1
second: two
third:
- 1
- 2
- 3
</pre>
```

6.9 FormHelper

Form helpers are designed to make working with models much easier compared to using just standard HTML elements by providing a set of methods for creating forms based on your models. This helper generates the HTML for forms, providing a method for each sort of input (e.g., text, password, select, and so on). When the form is submitted (i.e., when the user hits the submit button or form.submit is called via JavaScript), the form inputs will be bundled into the params object and passed back to the controller.

There are two types of form helpers: those that specifically work with model attributes and those that don't. This helper deals with those that work with model attributes; to see an example of form helpers that don't work with model attributes, check the ActionView::Helpers::FormTagHelper documentation.

The core method of this helper, form_for, gives you the ability to create a form for a model instance; for example, let's say that you have a model Person and want to create a new instance of it:

```
# Note: a @person variable will have been created in the controller (e.g. @person =
Person.new)
<%= form_for @person, url: {action: "create"} do |f| %>
  <%= f.text_field :first_name %>
  <%= f.text_field :last_name %>
  <%= submit_tag 'Create' %>
<% end %>
```

The HTML generated for this would be:

```
<form action="/people/create" method="post">
  <input id="person_first_name" name="person[first_name]" type="text" />
  <input id="person_last_name" name="person[last_name]" type="text" />
  <input name="commit" type="submit" value="Create" />
</form>
```

The params object created when this form is submitted would look like:

```
{"action" => "create", "controller" => "people", "person" => {"first_name" =>
"William", "last_name" => "Smith"}}
```

The params hash has a nested person value, which can therefore be accessed with params[:person] in the controller.

6.9.1 check_box Returns a checkbox tag tailored for accessing a specified attribute.

```
# Let's say that @article.validated? is 1:
check_box("article", "validated")
# => <input type="checkbox" id="article_validated" name="article[validated]" value=
"1" />
#    <input name="article[validated]" type="hidden" value="0" />
```

6.9.2 fields_for Creates a scope around a specific model object like form_for, but doesn't create the form tags themselves. This makes fields_for suitable for specifying additional model objects in the same form:

```
<%= form_for @person, url: {action: "update"} do |person_form| %>
  First name: <%= person_form.text_field :first_name %>
  Last name : <%= person_form.text_field :last_name %>

  <%= fields_for @person.permission do |permission_fields| %>
    Admin?  : <%= permission_fields.check_box :admin %>
  <% end %>
<% end %>
```

6.9.3 file_field Returns a file upload input tag tailored for accessing a specified attribute.

```
file_field(:user, :avatar)
# => <input type="file" id="user_avatar" name="user[avatar]" />
```

6.9.4 form_for Creates a form and a scope around a specific model object that is used as a base for questioning about values for the fields.

```
<%= form_for @article do |f| %>
  <%= f.label :title, 'Title' %>:
  <%= f.text_field :title %><br>
  <%= f.label :body, 'Body' %>:
  <%= f.text_area :body %><br>
<% end %>
```

6.9.5 hidden_field Returns a hidden input tag tailored for accessing a specified attribute.

```
hidden_field(:user, :token)
# => <input type="hidden" id="user_token" name="user[token]" value="#{@user.token}" /
>
```

6.9.6 label Returns a label tag tailored for labelling an input field for a specified attribute.

```
label(:article, :title)
# => <label for="article_title">Title</label>
```

6.9.7 password_field Returns an input tag of the "password" type tailored for accessing a specified attribute.

```
password_field(:login, :pass)
# => <input type="text" id="login_pass" name="login[pass]" value="#{@login.pass}" /
>
```

6.9.8 radio_button Returns a radio button tag for accessing a specified attribute.

```
# Let's say that @article.category returns "rails":
radio_button("article", "category", "rails")
radio_button("article", "category", "java")
# => <input type="radio" id="article_category_rails" name="article[category]" value=
"rails" checked="checked" />
#    <input type="radio" id="article_category_java" name="article[category]" value=
"java" />
```

6.9.9 text_area Returns a textarea opening and closing tag set tailored for accessing a specified attribute.

```
text_area(:comment, :text, size: "20x30")
# => <textarea cols="20" rows="30" id="comment_text" name="comment[text]">
#      #{@comment.text}
#    </textarea>
```

6.9.10 text_field Returns an input tag of the "text" type tailored for accessing a specified attribute.

```
text_field(:article, :title)
# => <input type="text" id="article_title" name="article[title]" value="#
{@article.title}" />
```

6.9.11 email_field Returns an input tag of the "email" type tailored for accessing a specified attribute.

```
email_field(:user, :email)
# => <input type="email" id="user_email" name="user[email]" value="#{@user.email}" /
>
```

6.9.12 url_field Returns an input tag of the "url" type tailored for accessing a specified attribute.

```
url_field(:user, :url)
# => <input type="url" id="user_url" name="user[url]" value="#{@user.url}" />
```

6.10 FormOptionsHelper

Provides a number of methods for turning different kinds of containers into a set of option tags.

6.10.1 collection_select Returns `select` and `option` tags for the collection of existing return values of `method` for `object`'s class.

Example object structure for use with this method:

```
class Article < ActiveRecord::Base
  belongs_to :author
end

class Author < ActiveRecord::Base
  has_many :articles
  def name_with_initial
    "#{first_name.first}. #{last_name}"
  end
end
```

Sample usage (selecting the associated Author for an instance of Article, `@article`):

```
collection_select(:article, :author_id, Author.all, :id, :name_with_initial, {prompt:
true})
```

If `@article.author_id` is 1, this would return:

```
<select name="article[author_id]">
  <option value="">Please select</option>
  <option value="1" selected="selected">D. Heinemeier Hansson</option>
  <option value="2">D. Thomas</option>
  <option value="3">M. Clark</option>
</select>
```

6.10.2 collection_radio_buttons Returns `radio_button` tags for the collection of existing return values of `method` for `object`'s class.

Example object structure for use with this method:

```
class Article < ActiveRecord::Base
  belongs_to :author
end

class Author < ActiveRecord::Base
  has_many :articles
  def name_with_initial
    "#{first_name.first}. #{last_name}"
  end
end
```

Sample usage (selecting the associated Author for an instance of Article, `@article`):

```
collection_radio_buttons(:article, :author_id, Author.all, :id, :name_with_initial)
```

If `@article.author_id` is 1, this would return:

```
<input id="article_author_id_1" name="article[author_id]" type="radio" value="1"
checked="checked" />
<label for="article_author_id_1">D. Heinemeier Hansson</label>
<input id="article_author_id_2" name="article[author_id]" type="radio" value="2" />
<label for="article_author_id_2">D. Thomas</label>
<input id="article_author_id_3" name="article[author_id]" type="radio" value="3" />
<label for="article_author_id_3">M. Clark</label>
```

6.10.3 collection_check_boxes Returns `check_box` tags for the collection of existing return values of `method` for `object`'s class.

Example object structure for use with this method:

```
class Article < ActiveRecord::Base
  has_and_belongs_to_many :authors
end

class Author < ActiveRecord::Base
  has_and_belongs_to_many :articles
  def name_with_initial
    "#{first_name.first}. #{last_name}"
  end
end
```

Sample usage (selecting the associated Authors for an instance of Article, `@article`):

```
collection_check_boxes(:article, :author_ids, Author.all, :id, :name_with_initial)
```

If `@article.author_ids` is [1], this would return:

```
<input id="article_author_ids_1" name="article[author_ids][]" type="checkbox" value="1"
checked="checked" />
<label for="article_author_ids_1">D. Heinemeier Hansson</label>
<input id="article_author_ids_2" name="article[author_ids][]" type="checkbox" value="2"
/>
<label for="article_author_ids_2">D. Thomas</label>
<input id="article_author_ids_3" name="article[author_ids][]" type="checkbox" value="3"
/>
<label for="article_author_ids_3">M. Clark</label>
<input name="article[author_ids][]" type="hidden" value="" />
```

6.10.4 country_options_for_select Returns a string of option tags for pretty much any country in the world.

6.10.5 country_select Returns select and option tags for the given object and method, using country_options_for_select to generate the list of option tags.

6.10.6 option_groups_from_collection_for_select Returns a string of `option` tags, like `options_from_collection_for_select`, but groups them by `optgroup` tags based on the object relationships of the arguments.

Example object structure for use with this method:

```
class Continent < ActiveRecord::Base
  has_many :countries
  # attribs: id, name
end

class Country < ActiveRecord::Base
  belongs_to :continent
  # attribs: id, name, continent_id
end
```

Sample usage:

```
option_groups_from_collection_for_select(@continents, :countries, :name, :id, :name, 3)
```

Possible output:

```
<optgroup label="Africa">
  <option value="1">Egypt</option>
  <option value="4">Rwanda</option>
  ...
</optgroup>
<optgroup label="Asia">
  <option value="3" selected="selected">China</option>
  <option value="12">India</option>
  <option value="5">Japan</option>
  ...
</optgroup>
```

Note: Only the `optgroup` and `option` tags are returned, so you still have to wrap the output in an appropriate `select` tag.

6.10.7 options_for_select Accepts a container (hash, array, enumerable, your type) and returns a string of option tags.

```
options_for_select([ "VISA", "MasterCard" ])
# => <option>VISA</option> <option>MasterCard</option>
```

Note: Only the `option` tags are returned, you have to wrap this call in a regular HTML `select` tag.

6.10.8 options_from_collection_for_select Returns a string of option tags that have been compiled by iterating over the `collection` and assigning the result of a call to the `value_method` as the option value and the `text_method` as the option text.

```
# options_from_collection_for_select(collection, value_method, text_method, selected =
nil)
```

For example, imagine a loop iterating over each person in @project.people to generate an input tag:

```
options_from_collection_for_select(@project.people, "id", "name")
# => <option value="#{person.id}">#{person.name}</option>
```

Note: Only the `option` tags are returned, you have to wrap this call in a regular HTML `select` tag.

6.10.9 select Create a select tag and a series of contained option tags for the provided object and method.

Example:

```
select("article", "person_id", Person.all.collect {|p| [ p.name, p.id ] }, {include_blank:
true})
```

If `@article.person_id` is 1, this would become:

```
<select name="article[person_id]">
  <option value=""></option>
  <option value="1" selected="selected">David</option>
  <option value="2">Sam</option>
  <option value="3">Tobias</option>
</select>
```

6.10.10 time_zone_options_for_select Returns a string of option tags for pretty much any time zone in the world.

6.10.11 time_zone_select Returns select and option tags for the given object and method, using `time_zone_options_for_select` to generate the list of option tags.

```
time_zone_select( "user", "time_zone")
```

6.10.12 date_field Returns an input tag of the "date" type tailored for accessing a specified attribute.

```
date_field("user", "dob")
```

6.11 FormTagHelper

Provides a number of methods for creating form tags that don't rely on an Active Record object assigned to the template like FormHelper does. Instead, you provide the names and values manually.

6.11.1 check_box_tag Creates a check box form input tag.

```
check_box_tag 'accept'
# => <input id="accept" name="accept" type="checkbox" value="1" />
```

6.11.2 field_set_tag Creates a field set for grouping HTML form elements.

```
<%= field_set_tag do %>
  <p><%= text_field_tag 'name' %></p>
<% end %>
# => <fieldset><p><input id="name" name="name" type="text" /></
p></fieldset>
```

6.11.3 file_field_tag Creates a file upload field.

```
<%= form_tag({action:"post"}, multipart: true) do %>
  <label for="file">File to Upload</label> <%= file_field_tag "file" %>
  <%= submit_tag %>
<% end %>
```

Example output:

```
file_field_tag 'attachment'
# => <input id="attachment" name="attachment" type="file" />
```

6.11.4 form_tag Starts a form tag that points the action to an url configured with `url_for_options` just like `ActionController::Base#url_for`.

```
<%= form_tag '/articles' do %>
  <div><%= submit_tag 'Save' %></div>
<% end %>
# => <form action="/articles" method="post"><div><input type="submit"
name="submit" value="Save" /></div></form>
```

6.11.5 hidden_field_tag Creates a hidden form input field used to transmit data that would be lost due to HTTP's statelessness or data that should be hidden from the user.

```
hidden_field_tag 'token', 'VUBJKB23UIVI1UU1VOBVI@'
# => <input id="token" name="token" type="hidden" value="VUBJKB23UIVI1UU1VOBVI@" /
>
```

6.11.6 image_submit_tag Displays an image which when clicked will submit the form.

```
image_submit_tag("login.png")
# => <input src="/images/login.png" type="image" />
```

6.11.7 label_tag Creates a label field.

```
label_tag 'name'
# => <label for="name">Name</label>
```

6.11.8 password_field_tag Creates a password field, a masked text field that will hide the users input behind a mask character.

```
password_field_tag 'pass'
# => <input id="pass" name="pass" type="password" />
```

6.11.9 radio_button_tag Creates a radio button; use groups of radio buttons named the same to allow users to select from a group of options.

```
radio_button_tag 'gender', 'male'
# => <input id="gender_male" name="gender" type="radio" value="male" />
```

6.11.10 select_tag Creates a dropdown selection box.

```
select_tag "people", "<option>David</option>"
# => <select id="people" name="people"><option>David</option></
select>
```

6.11.11 submit_tag Creates a submit button with the text provided as the caption.

```
submit_tag "Publish this article"
# => <input name="commit" type="submit" value="Publish this article" />
```

6.11.12 text_area_tag Creates a text input area; use a textarea for longer text inputs such as blog posts or descriptions.

```
text_area_tag 'article'
# => <textarea id="article" name="article"></textarea>
```

6.11.13 text_field_tag Creates a standard text field; use these text fields to input smaller chunks of text like a username or a search query.

```
text_field_tag 'name'
# => <input id="name" name="name" type="text" />
```

6.11.14 email_field_tag Creates a standard input field of email type.

```
email_field_tag 'email'
# => <input id="email" name="email" type="email" />
```

6.11.15 url_field_tag Creates a standard input field of url type.

```
url_field_tag 'url'
# => <input id="url" name="url" type="url" />
```

6.11.16 date_field_tag Creates a standard input field of date type.

```
date_field_tag "dob"
# => <input id="dob" name="dob" type="date" />
```

6.12 JavaScriptHelper

Provides functionality for working with JavaScript in your views.

6.12.1 button_to_function Returns a button that'll trigger a JavaScript function using the onclick handler. Examples:

```
button_to_function "Greeting", "alert('Hello world!')"
button_to_function "Delete", "if (confirm('Really?')) do_delete()"
button_to_function "Details" do |page|
  page[:details].visual_effect :toggle_slide
end
```

6.12.2 define_javascript_functions Includes the Action Pack JavaScript libraries inside a single `script` tag.

6.12.3 escape_javascript Escape carrier returns and single and double quotes for JavaScript segments.

6.12.4 javascript_tag Returns a JavaScript tag wrapping the provided code.

```
javascript_tag "alert('All is good')"
```

```
<script>
//<![CDATA[
alert('All is good')
//]]>
</script>
```

6.12.5 link_to_function Returns a link that will trigger a JavaScript function using the onclick handler and return false after the fact.

```
link_to_function "Greeting", "alert('Hello world!')"
# => <a onclick="alert('Hello world!'); return false;" href="#">Greeting</a>
```

6.13 NumberHelper

Provides methods for converting numbers into formatted strings. Methods are provided for phone numbers, currency, percentage, precision, positional notation, and file size.

6.13.1 number_to_currency Formats a number into a currency string (e.g., $13.65).

```
number_to_currency(1234567890.50) # => $1,234,567,890.50
```

6.13.2 number_to_human_size Formats the bytes in size into a more understandable representation; useful for reporting file sizes to users.

```
number_to_human_size(1234)          # => 1.2 KB
number_to_human_size(1234567)       # => 1.2 MB
```

6.13.3 number_to_percentage Formats a number as a percentage string.

```
number_to_percentage(100, precision: 0)        # => 100%
```

6.13.4 number_to_phone Formats a number into a US phone number.

```
number_to_phone(1235551234) # => 123-555-1234
```

6.13.5 number_with_delimiter Formats a number with grouped thousands using a delimiter.

```
number_with_delimiter(12345678) # => 12,345,678
```

6.13.6 number_with_precision Formats a number with the specified level of `precision`, which defaults to 3.

```
number_with_precision(111.2345)     # => 111.235
number_with_precision(111.2345, 2)  # => 111.23
```

6.14 SanitizeHelper

The SanitizeHelper module provides a set of methods for scrubbing text of undesired HTML elements.

6.14.1 sanitize This sanitize helper will HTML encode all tags and strip all attributes that aren't specifically allowed.

```
sanitize @article.body
```

If either the :attributes or :tags options are passed, only the mentioned tags and attributes are allowed and nothing else.

```
sanitize @article.body, tags: %w(table tr td), attributes: %w(id class style)
```

To change defaults for multiple uses, for example adding table tags to the default:

```
class Application < Rails::Application
  config.action_view.sanitized_allowed_tags = 'table', 'tr', 'td'
end
```

6.14.2 sanitize_css(style) Sanitizes a block of CSS code.

6.14.3 strip_links(html) Strips all link tags from text leaving just the link text.

```
strip_links("<a href="http://rubyonrails.org">Ruby on Rails</a>")
# => Ruby on Rails

strip_links("emails to <a href="mailto:me@email.com">me@email.com</a>.")
# => emails to me@email.com.

strip_links('Blog: <a href="http://myblog.com/">Visit</a>.')
# => Blog: Visit.
```

6.14.4 strip_tags(html) Strips all HTML tags from the html, including comments. This uses the html-scanner tokenizer and so its HTML parsing ability is limited by that of html-scanner.

```
strip_tags("Strip <i>these</i> tags!")
# => Strip these tags!

strip_tags("<b>Bold</b> no more!  <a href='more.html'>See more</a>
")
# => Bold no more!  See more
```

NB: The output may still contain unescaped '<', '>', '&' characters and confuse browsers.

6.15 CsrfHelper

Returns meta tags "csrf-param" and "csrf-token" with the name of the cross-site request forgery protection parameter and token, respectively.

```
<%= csrf_meta_tags %>
```

Regular forms generate hidden fields so they do not use these tags. More details can be found in the Rails Security Guide.

7 Localized Views

Action View has the ability render different templates depending on the current locale.

For example, suppose you have a `ArticlesController` with a show action. By default, calling this action will render `app/views/articles/show.html.erb`. But if you set `I18n.locale = :de`, then `app/views/articles/show.de.html.erb` will be rendered instead. If the localized template isn't present, the undecorated version will be used. This means you're not required to provide localized views for all cases, but they will be preferred and used if available.

You can use the same technique to localize the rescue files in your public directory. For example, setting `I18n.locale = :de` and creating `public/500.de.html` and `public/404.de.html` would allow you to have localized rescue pages.

Since Rails doesn't restrict the symbols that you use to set I18n.locale, you can leverage this system to display different content depending on anything you like. For example, suppose you have some "expert" users that should see different pages from "normal" users. You could add the following to `app/controllers/application.rb`:

```
before_action :set_expert_locale

def set_expert_locale
  I18n.locale = :expert if current_user.expert?
end
```

Then you could create special views like `app/views/articles/show.expert.html.erb` that would only be displayed to expert users.

You can read more about the Rails Internationalization (I18n) API here.

8 Feedback

You're encouraged to help improve the quality of this guide.

Please contribute if you see any typos or factual errors. To get started, you can read our documentation contributions section.

You may also find incomplete content, or stuff that is not up to date. Please do add any missing documentation for master. Make sure to check Edge Guides first to verify if the issues are already fixed or not on the master branch. Check the Ruby on Rails Guides Guidelines for style and conventions.

If for whatever reason you spot something to fix but cannot patch it yourself, please open an issue.

And last but not least, any kind of discussion regarding Ruby on Rails documentation is very welcome in the rubyonrails-docs mailing list.