Action View Helpers

After reading this guide, you will know:

- How to format dates, strings and numbers
- How to link to images, videos, stylesheets, etc…
- How to sanitize content
- How to localize content

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.

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

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

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
```

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
```

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" />
```

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>
```

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
```

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
```

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" rel="stylesheet" />
```

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
```

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
```

AtomFeedHelper

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
```

BenchmarkHelper

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.

CacheHelper

cache A method for caching fragments of a view rather than an entire action or page. This technique is useful for 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 `AbstractController::Caching::Fragments` for more information.

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

CaptureHelper

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>
```

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 %>
```

DateHelper

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
```

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
```

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>
```

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.

You can learn more about form helpers in the Action View Form Helpers Guide.

JavaScriptHelper

Provides functionality for working with JavaScript in your views.

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

javascript_tag Returns a JavaScript tag wrapping the provided code.

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

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

NumberHelper

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

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

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

number_to_human Pretty prints (formats and approximates) a number so it is more readable by users; useful for numbers that can get very large.

```
number_to_human(1234)    # => 1.23 Thousand
number_to_human(1234567) # => 1.23 Million
```

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)    # => 1.21 KB
number_to_human_size(1234567) # => 1.18 MB
```

number_to_percentage Formats a number as a percentage string.

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

number_to_phone Formats a number into a phone number (US by default).

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

number_with_delimiter Formats a number with grouped thousands using a delimiter.

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

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, precision: 2) # => 111.23
```

SanitizeHelper

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

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 attributes and tags 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
```

sanitize_css(style) Sanitizes a block of CSS code.

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

```
strip_links('<a href="https://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.
```

strip_tags(html) Strips all HTML tags from the html, including comments. This functionality is powered by the rails-html-sanitizer gem.

```
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.

UrlHelper

Provides methods to make links and get URLs that depend on the routing subsystem.

url_for Returns the URL for the set of `options` provided.

Examples

```
url_for @profile
# => /profiles/1

url_for [ @hotel, @booking, page: 2, line: 3 ]
# => /hotels/1/bookings/1?line=3&page=2
```

link_to Links to a URL derived from `url_for` under the hood. Primarily used to create RESTful resource links, which for this example, boils down to when passing models to `link_to`.

Examples

```
link_to "Profile", @profile
# => <a href="/profiles/1">Profile</a>
```

You can use a block as well if your link target can't fit in the name parameter. ERB example:

```
<%= link_to @profile do %>
  <strong><%= @profile.name %></strong> -- <span>Check it out!</span>
<% end %>
```

would output:

```
<a href="/profiles/1">
  <strong>David</strong> -- <span>Check it out!</span>
</a>
```

See the API Documentation for more information

button_to Generates a form that submits to the passed URL. The form has a submit button with the value of the `name`.

Examples

```
<%= button_to "Sign in", sign_in_path %>
```

would roughly output something like:

```
<form method="post" action="/sessions" class="button_to">
  <input type="submit" value="Sign in" />
</form>
```

See the API Documentation for more information

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 %>
```

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