

 **AKovtunov** on 30 Mar Changed old-style hashes to the new-style in readme and gemfiles33 contributors  and others

336 lines (284 sloc) 16.149 kb

Raw Blame History   

Wicked PDF

A PDF generation plugin for Ruby on Rails

Wicked PDF uses the shell utility [wkhtmltopdf](#) to serve a PDF file to a user from HTML. In other words, rather than dealing with a PDF generation DSL of some sort, you simply write an HTML view as you would normally, then let Wicked PDF take care of the hard stuff.

Wicked PDF has been verified to work on Ruby versions 1.8.7 through 2.1; Rails 2 through 4.1

Installation

Add this to your Gemfile and run `bundle install`:

```
gem 'wicked_pdf'
```

 InsertIntoFile and RootFile

Then create the initializer with

```
rails generate wicked_pdf
```

 RootFile

You may also need to add

```
Mime::Type.register "application/pdf", :pdf
```

 InsertIntoFile

to `config/initializers/mime_types.rb` in older versions of Rails.

Because `wicked_pdf` is a wrapper for [wkhtmltopdf](#), you'll need to install that, too.

The simplest way to install all of the binaries (Linux, OSX, Windows) is through the gem [wkhtmltopdf-binary](#). To install that, add a second gem

```
gem 'wkhtmltopdf-binary'
```

 InsertIntoFile

To your Gemfile and run `bundle install`.

If your `wkhtmltopdf` executable is not on your webserver's path, you can configure it in an initializer:

```
WickedPdf.config = {  
  exe_path: '/usr/local/bin/wkhtmltopdf'  
}
```

 InsertIntoFile

For more information about `wkhtmltopdf`, see the project's [homepage](#).

Basic Usage

```
class ThingsController < ApplicationController
  def show
    respond_to do |format|
      format.html
      format.pdf do
        render pdf: "file_name" # Excluding ".pdf" extension.
      end
    end
  end
end
```



InsertIntoFile

Usage Conditions - Important!

The wkhtmltopdf binary is run outside of your Rails application; therefore, your normal layouts will not work. If you plan to use any CSS, Javascript, or image files, you must modify your layout so that you provide an absolute reference to these files. The best option for Rails without the asset pipeline is to use the `wicked_pdf_stylesheet_link_tag`, `wicked_pdf_image_tag`, and `wicked_pdf_javascript_include_tag` helpers or to go straight to a CDN (Content Delivery Network) for popular libraries such as jQuery.

wicked_pdf helpers

```
<!doctype html>
<html>
  <head>
    <meta charset='utf-8' />
    <%= wicked_pdf_stylesheet_link_tag "pdf" -%>
    <%= wicked_pdf_javascript_include_tag "number_pages" -%>
  </head>
  <body onload='number_pages'>
    <div id="header">
      <%= wicked_pdf_image_tag 'mysite.jpg' -%>
    </div>
    <div id="content">
      <%= yield -%>
    </div>
  </body>
</html>
```



InsertIntoFile

CDN reference

In this case, you can use that standard Rails helpers and point to the current CDN for whichever framework you are using. For jQuery, it would look something like this, given the current versions at the time of this writing.

```
<!doctype html>
<html>
  <head>
    <%= javascript_include_tag "http://code.jquery.com/jquery-1.10.0.min.js" -%>
    <%= javascript_include_tag "http://code.jquery.com/ui/1.10.3/jquery-ui.min.js" -%>
```




InsertIntoFile

Asset pipeline usage

The way to handle this for the asset pipeline on Heroku is to include these files in your asset precompile list, as follows:

```
config.assets.precompile += ['blueprint/screen.css', 'pdf.css', 'jquery.ui.datepicker.js', 'pdf.js', ...etc...]
```



InsertIntoFile

Advanced Usage with all available options

```
class ThingsController < ApplicationController
  def show
    respond_to do |format|
```

```

format.html
format.pdf do
  render pdf:
    disposition: 'file_name',
    template: 'attachment', # default 'inline'
    file: 'things/show.pdf.erb',
    layout: "#{Rails.root}/files/foo.erb"
    wkhtmltopdf: 'pdf.html', # use 'pdf.html' for a pdf.html.erb
    show_as_html: '/usr/local/bin/wkhtmltopdf', # path to binary
    orientation: params[:debug].present?, # allow debugging based on url param
    page_size: 'Landscape', # default Portrait
    page_height: 'A4, Letter, ...', # default A4
    page_width: NUMBER,
    save_to_file: NUMBER,
    save_only: Rails.root.join('pdfs', "#{filename}.pdf"),
    proxy: false, # depends on :save_to_file being set
    basic_auth: 'TEXT',
    username: false # when true username & password are
    password: 'TEXT',
    title: 'Alternate Title', # otherwise first page title is used
    cover: 'URL, Pathname, or raw HTML string',
    dpi: 'dpi',
    encoding: 'TEXT',
    user_style_sheet: 'URL',
    cookie: ['_session_id SESSION_ID'], # could be an array or a single string
    post: ['query QUERY_PARAM'], # could be an array or a single string
    redirect_delay: NUMBER,
    javascript_delay: NUMBER,
    image_quality: NUMBER,
    no_pdf_compression: true,
    zoom: FLOAT,
    page_offset: NUMBER,
    book: true,
    default_header: true,
    disable_javascript: false,
    grayscale: true,
    lowquality: true,
    enable_plugins: true,
    disable_internal_links: true,
    disable_external_links: true,
    print_media_type: true,
    disable_smart_shrinking: true,
    use_xserver: true,
    no_background: true,
    viewport_size: 'TEXT', # available only with use_xserver or ps
    extra: '', # directly inserted into the command to
    outline: {
      outline: true,
      outline_depth: LEVEL },
    margin: {
      top: SIZE, # default 10 (mm)
      bottom: SIZE,
      left: SIZE,
      right: SIZE },
    header: {
      html: {
        template: 'users/header.pdf.erb', # use :template OR :url
        layout: 'pdf_plain.html', # optional, use 'pdf_plain.html'
        url: 'www.example.com',
        locals: { foo: @bar }},
        center: 'TEXT',
        font_name: 'NAME',
        font_size: SIZE,
        left: 'TEXT',
        right: 'TEXT',
        spacing: REAL,
        line: true,
        content: 'HTML CONTENT ALREADY RENDERED'}, # optionally you can pass plain
      footer: {
        html: {
          template: 'shared/footer.pdf.erb', # use :template OR :url
          layout: 'pdf_plain.html', # optional, use 'pdf_plain.html' for a pc
          url: 'www.example.com',
          locals: { foo: @bar }},
          center: 'TEXT',
          font_name: 'NAME',
          font_size: SIZE,
          left: 'TEXT',
          right: 'TEXT',
          spacing: REAL,

```

```

      line:          true,
      content:       'HTML CONTENT ALREADY RENDERED'}, # optionally you can pass plain
toc:    { font_name:  "NAME",
          depth:      LEVEL,
          header_text: "TEXT",
          header_fs:   SIZE,
          text_size_shrink: 0.8,
          l1_font_size: SIZE,
          l2_font_size: SIZE,
          l3_font_size: SIZE,
          l4_font_size: SIZE,
          l5_font_size: SIZE,
          l6_font_size: SIZE,
          l7_font_size: SIZE,
          level_indentation: NUM,
          l1_indentation: NUM,
          l2_indentation: NUM,
          l3_indentation: NUM,
          l4_indentation: NUM,
          l5_indentation: NUM,
          l6_indentation: NUM,
          l7_indentation: NUM,
          no_dots:      true,
          disable_dotted_lines: true,
          disable_links: true,
          disable_toc_links: true,
          disable_back_links: true,
          xsl_style_sheet: 'file.xsl'} # optional XSLT stylesheet to use for styling table of
    end
  end
end
end

```



By default, it will render without a layout (layout: false) and the template for the current controller and action.

Super Advanced Usage

If you need to just create a pdf and not display it:

```

# create a pdf from a string
pdf = WickedPdf.new.pdf_from_string('<h1>Hello There!</h1>')


# create a pdf file from a html file without converting it to string
# Path must be absolute path
pdf = WickedPdf.new.pdf_from_html_file('/your/absolute/path/here')

# create a pdf from string using templates, layouts and content option for header or footer
WickedPdf.new.pdf_from_string(
  render_to_string('templates/pdf.html.erb', layout: 'pdfs/layout_pdf'),
  footer: {
    content: render_to_string(layout: 'pdfs/layout_pdf')
  }
)

# or from your controller, using views & templates and all wicked_pdf options as normal
pdf = render_to_string pdf: "some_file_name", template: "templates/pdf.html.erb", encoding: "UTF-8"

# then save to a file
save_path = Rails.root.join('pdfs', 'filename.pdf')
File.open(save_path, 'wb') do |file|
  file << pdf
end

```



If you need to display utf encoded characters, add this to your pdf views or layouts:

```
<meta charset="utf-8" />
```

Page Numbering

A bit of javascript can help you number your pages. Create a template or header/footer file with this:

```
<html>
<head>
  <script>
    function number_pages() {
      var vars={};
      var x=document.location.search.substring(1).split('&');
      for(var i in x) {var z=x[i].split('=');vars[z[0]] = unescape(z[1]);}
      var x=['frompage','topage','page','webpage','section','subsection','subsubsection'];
      for(var i in x) {
        var y = document.getElementsByClassName(x[i]);
        for(var j=0; j<y.length; ++j) y[j].textContent = vars[x[i]];
      }
    }
  </script>
</head>
<body onload="number_pages()">
  Page <span class="page"></span> of <span class="topage"></span>
</body>
</html>
```


 InsertIntoFile

Anything with a class listed in "var x" above will be auto-filled at render time.

If you do not have explicit page breaks (and therefore do not have any "page" class), you can also use wkhtmltopdf's built in page number generation by setting one of the headers to "[page]":

```
render pdf: 'filename', header: { right: '[page] of [topage]' }
```

Configuration

You can put your default configuration, applied to all pdf's at "wicked_pdf.rb" initializer.

Rack Middleware

If you would like to have WickedPdf automatically generate PDF views for all (or nearly all) pages by appending .pdf to the URL, add the following to your Rails app:

```
# in application.rb (Rails3) or environment.rb (Rails2)
require 'wicked_pdf'
config.middleware.use WickedPdf::Middleware
```


 InsertIntoFile

If you want to turn on or off the middleware for certain urls, use the :only or :except conditions like so:

```
# conditions can be plain strings or regular expressions, and you can supply only one or an array
config.middleware.use WickedPdf::Middleware, {}, only: '/invoice'
config.middleware.use WickedPdf::Middleware, {}, except: [ %r[^\d/admin], '/secret', %r[^\d/people/] ]
```


 InsertIntoFile

If you use the standard render pdf: 'some_pdf' in your app, you will want to exclude those actions from the middleware.

Further Reading

Andreas Happe's post [Generating PDFs from Ruby on Rails](#)

JESii's post [WickedPDF, wkhtmltopdf, and Heroku...a tricky combination](#)

StackOverflow [questions with the tag "wicked-pdf"](#)

Debugging

Now you can use a debug param on the URL that shows you the content of the pdf in plain html to design it faster.

First of all you must configure the render parameter "show_as_html: params[:debug]" and then just use it like normally

but adding "debug=1" as a param:

`http://localhost:3001/CONTROLLER/X.pdf?debug=1`

However, the `wicked_pdf_*` helpers will use `file:///` paths for assets when using `:show_as_html`, and your browser's cross-domain safety feature will kick in, and not render them. To get around this, you can load your assets like so in your templates:

```
<%= params[:debug].present? ? image_tag('foo') : wicked_pdf_image_tag('foo') %>
```

Gotchas

If one image from your HTML cannot be found (relative or wrong path for ie), others images with right paths **may not** be displayed in the output PDF as well (it seems to be an issue with wkhtmltopdf).

Inspiration

You may have noticed: this plugin is heavily inspired by the PrinceXML plugin [princely](#). PrinceXML's cost was prohibitive for me. So, with a little help from some friends (thanks [jqr](#)), I tracked down wkhtmltopdf, and here we are.

Contributing

1. Fork it
2. Create your feature branch (`git checkout -b my-new-feature`)
3. Commit your changes (`git commit -am 'Add some feature'`)
4. Push to the branch (`git push origin my-new-feature`)
5. Create new Pull Request

Awesome Peoples

Also, thanks to [unixmonkey](#), [galdomedia](#), [jcrisp](#), [lleirborras](#), [tiennou](#), and everyone else for all their hard work and patience with my delays in merging in their enhancements.

