# Print pages to PDF files

Install print plugins. Add cover page and customize styles for printing. Automatically export to PDF files when building the site.

#python #mkdocs

---

Last update: April 27, 2021

# Table of Content

# 1. Print plugin

The MkDocs PDF with JS Plugin[1] exports documentation as PDF with rendered JavaScript content. This is very useful if documents have mermaid diagrams. A download button will be added to the top of the documentation.

For executing the JavaScript code ChromeDriver is used. So it is nesseccary to:

1. Install Chrome
2. Download ChromeDriver
3. Add the ChromeDriver to OS user's `PATH`

Install the plugin:

```
pip install -U git+https://github.com/vuquangtrong/mkdocs-pdf-with-js-plugin.git
```

*Install original plugin with `pip install mkdocs-pdf-with-js-plugin` if don't need a customized version*

Enable the plugin:

```
plugins:
    - search # built-in search must be always activated
    - pdf-with-js
```

While building `mkdocs build` or serving `mkdocs serve` the documentation, the pdfs get generated. They are stored in *site_dir/pdfs*.

⚠ This is a modification in the plugin MkDocs PDF with JS Plugin, if it's installed, it is ready to use without any further changes showing in below sections

# 2. Add download button

Create an element to contain the download button at the beginning of the document content in the *base.html* template:

```
<div class="md-content" data-md-component="content">
    <article class="md-content__inner md-typeset">
    {% block content %}
        {% if not page.is_homepage %}
            <div class="btn-actions"></div>
        {% endif %}
        {{ page.content }}
    ...
    {% endblock %}
    </article>
</div>
```

The plugin will find the `<div class="btn-actions">` element to insert a button:

*.venv\Lib\site-packages\pdf_with_js\printer.py*

```
def _add_link(self, soup, page_paths):
    icon = BeautifulSoup(''
        '<span class="twemoji">'
            '<svg viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg">'
                '<path d="M5 20h14v-2H5m14-9h-4V3H9v6H5l7 7 7-7z"></path>'
            '</svg>'
        '</span>',
        'html.parser')
    text = "PDF"

    btn = soup.new_tag("a", href=page_paths["relpath"])
    btn.append(icon)
    btn.append(text)
    btn['class'] = 'md-button'

    div = soup.new_tag("div")
    div['id'] = 'btn-download'
    div.append(btn)

    bar = soup.find("div", {"class" : "btn-actions"})
    if bar:
        bar.insert(0, div)

    return soup
```

## 3. Add header and footer

The command sent to ChromeDriver to print a page is *Page.printToPDF*, read more at Chrome DevTools Protocol - printToPDF.

Some important params:

**landscape** : *boolean*

Paper orientation. Defaults to *false*.

**displayHeaderFooter** : *boolean*

Display header and footer. Defaults to *false*.

**headerTemplate**: : *string*

HTML template for the print header. Should be valid HTML markup with following classes used to inject printing values into them:

- *date*: formatted print date

- *title*: document title
- *url*: document location
- *pageNumber*: current page number
- *totalPages*: total pages in the document

For example, `<span class=title></span>` would generate span containing the title.

**footerTemplate** : *string*

HTML template for the print footer. Should use the same format as the headerTemplate.

Initialize default value for those params:

```
def __init__(self):
    self.pages = []
    self.filenameUseFullTitle = False
    self.displayHeaderFooter = True
    self.headerTemplate =   '<div style="font-size:8px; margin:auto;">' \
                            '<span class=title></span>' \
                            '</div>'
    self.footerTemplate=    '<div style="font-size:8px; margin:auto;">' \
                            'Page <span class="pageNumber"></span> of <span
class="totalPages"></span>' \
                            '</div>'
```

then create print options in json format:

```
def _get_print_options(self):
    return {
        'landscape': False,
        'displayHeaderFooter': self.displayHeaderFooter,
        'footerTemplate': self.footerTemplate,
        'headerTemplate': self.headerTemplate,
        'printBackground': True,
        'preferCSSPageSize': True,
    }
```

and use it when calling the print command:

```
def print_to_pdf(self, driver, page):
    driver.get(page["url"])
    result = self._send_devtools_command(driver, "Page.printToPDF",
self._get_print_options())
    self._write_file(result['data'], page["pdf_file"])
```

# 4. Add plugin config option

To allow user to change the print option in the project config file *mkdocs.yml*, add the config fields into the *plugin.py* file.

```
class PdfWithJS(BasePlugin):
    config_scheme = (
        ('enable', config_options.Type(bool, default=True)),
        ('filename_use_full_title', config_options.Type(bool, default=False)),
        ('display_header_footer', config_options.Type(bool, default=False)),
        ('header_template', config_options.Type(str, default='')),
        ('footer_template', config_options.Type(str, default='')),
    )
```

when the MkDocs engine calls to *on_config()* function in this plugin, save the user's configs as below:

```
def on_config(self, config, **kwargs):
    self.enabled = self.config['enable']
    self.printer.set_config (
        self.config['filename_use_full_title'],
        self.config['display_header_footer'],
        self.config['header_template'],
        self.config['footer_template']
        )

    return config
```

# 5. Printing styles

When printing, many elements should not show or have different style to fit the paper size or match with overall style on paper. Change the look of elements by adding styles in the additional stylesheet *assets|extra.css*.

extra.css

## 5.1. Cover page

The first page should show the topic and a short description of the post.

Create an element named *page-cover* in the *base.html* template:

```
    {% block content %}
        <div class="page-cover">
            {% if not "\x3ch1" in page.content %}
                {% if page and page.meta and page.meta.title_full %}
                    <h1 class="page-title">{{ page.meta.title_full |
d(config.site_name, true) }}</h1>
                {% elif page and page.meta and page.meta.title %}
                    <h1 class="page-title">{{ page.meta.title |
```

```
d(config.site_name, true) }}</h1>
                {% else %}
                    <h1 class="page-title">{{ config.site_name }}</h1>
                {% endif %}
                {% if page and page.meta and page.meta.description %}
                    <p class="page-description">{{ page.meta.description }}</p>
                {% endif %}
                {% if page and page.meta and page.meta.tags %}
                    <p>
                    {% for tag in page.meta.tags %}
                        <a class="tag" href="{{ config.site_url }}tags/#{{tag}}">
                        <span class="tag-name" style="color:{{ random_color()
}};">
                            #{{ tag }}
                        </span>
                        </a>
                    {% endfor %}
                    </p>
                {% endif %}
            {% endif %}
        </div>
        {% if not page.is_homepage %}
            <div class="btn-actions screen-only"></div>
            <div class="toc print-only">
                <h2>Table of Content</h2>
                {% include "partials/toc.html" %}
            </div>
        {% endif %}
        {{ page.content }}
```

The page cover will use 100% height of the paper and align its content vertically:

```
@media print {
    .md-typeset .page-cover {
        height: 100vh;
        display: flex;
        flex-direction: column;
        justify-content: center;
    }
}
```

## Footnotes:

1. originally developed by smaxtec ↩