

# Customize Plugin

## Documentation

Revision 0.1.8

*Duodu Randy*



# Table of Contents

---

1. Custom cover page	3
1.1. Using custom cover template	3
2. Adjusting the output	5
2.1. Changing the orientation of a page	6
2.1.1. Example	7

# 1. Custom cover page

You can create a custom cover page using a [jinja2](#) or html file.

## Note

You should store the custom cover template file in the directory used for the [custom\\_template\\_path](#).

The plugin provides the following variables which you can use in your custom Jinja template:

- cover\_title
- cover\_subtitle
- cover\_image
- author
- author\_logo
- copyright
- disclaimer
- site\_url
- revision
- custom variables from the [extra:](#) setting in your `mkdocs.yml`
- and all the options you provide under [local pdf metadata](#) of a Markdown file.

Using [jinja2](#) syntax, you can access all the data above. E.g. use `{{ author }}` to get the value for the [author](#) option:

```
plugins:  
  - pdf-generate:  
      author: Duodu Randy
```

## 1.1. Using custom cover template

You can specify the cover page to use for your PDF by following these steps:

### Step 1

Set the [custom\\_template\\_path](#) option for the plugin to the directory you want to store the cover template file.

```
plugins:  
  - pdf-generate:  
      custom_template_path: TEMPLATES_PATH
```

## Step 2

Under the directory you set as `custom_template_path`, create a custom template file. The custom template's filename can either be `cover` or the [document type](#).

E.g. `cover.html`, `cover.html.j2`, or if the document type is `manual` then `manual.html`, `manual.html.j2`.

In the cover template file, write your preferred template syntax into it.

*Example of a cover template file using Jinja2 syntax:*

```
<article id="doc-cover">
    {% if cover_image is defined %}
        <div class="wrapper upper">
            <div class="logo" style="background-image: url('{{ cover_image | to_url }}');"></div>
        </div>
    {% else %}
        <div class="wrapper"></div>
    {% endif %}
    <div class="wrapper">
        <h1>{{ cover_title | e }}</h1>
        <h2>{{ cover_subtitle | e }}</h2>
        {% if revision %}
            <h3>Revision {{ revision | e }}</h3>
        {% endif %}
    </div>
    <div class="properties">
        <address>
            {% if author %}
                <p id="author">{{ author | e }}</p>
            {% endif %}
            <a href="{{ site_url }}" id="project_logo" title="Resource Centre">
                
            </a>
        </address>
    </div>
    <div class="reserved_rights">
        <address>
            {% if copyright %}
                <p id="copyright">{{ copyright | e }}</p>
            {% endif %}
            {% if disclaimer %}
                <p id="disclaimer">{{ disclaimer | e }}</p>
            {% endif %}
        </address>
    </div>
</article>
```

## Step 3

Save the file changes and rebuild your MkDocs project.

## 2. Adjusting the output

The resulting PDF can be customized easily by adding a custom stylesheet such as the following:

```
@page {
  size: a4 portrait;
  margin: 25mm 10mm 25mm 10mm;
  counter-increment: page;
  font-family: "Roboto", "Helvetica Neue", Helvetica, Arial, sans-serif;
  white-space: pre;
  color: grey;
  @top-left {
    content: '© 2018 My Company';
  }
  @top-center {
    content: string(chapter);
  }
  @top-right {
    content: 'Page ' counter(page);
  }
}
```

For this to take effect, you need to create a `custom.css` file.



### Note

You should store the `custom.css` file in the directory used for the `custom_template_path`.

The plugin provides the following CSS variables and named strings which you can use in your `custom.css` file:

- `--title`
- `--subtitle`
- `--author`
- `--author-logo`
- `--copyright`
- `--type`
- `--site_url`
- `--revision`
- `--filename`
- `chapter`

Using the `var()` CSS function, you can access all the CSS variables provided by the plugin. E.g. use `var(--author)` to get the value for the `author` option.

You can also use the `string()` function to access the value of a named string. E.g. use `string(chapter)` to get the value for a chapter.

The custom CSS is appended to the MkDocs stylesheets so, you can override rules by using the `!important` CSS keyword but be cautious about it.

## 2.1. Changing the orientation of a page

The plugin allows you to change the orientation of a page to fit the content on that page.

For example, if you have a table on a page, and it is too wide to fit the current orientation used by the page, you can change the page orientation of the individual page by doing the following:

- Wrap the markdown content in a `div` html element. The `div` element should have a `markdown` attribute and a `class` attribute set to `"rotated-page"`. E.g. 


```
<div markdown class="rotated-page">PLACE CONTENT HERE</div>
```
- Create a `custom.css` file and set these CSS variables under a `:root {}` CSS selector:
  - `--base-page-orientation` - default page orientation to use and
  - `--rotated-page-orientation` - page orientation to use for rotated pages.E.g. 

```
:root {--base-page-orientation: a4 portrait; --rotated-page-orientation: a4 landscape;}
```



2.1.1. Example

In this example, we are going to change the page orientation for [this subsection](#).

 **Note**

Download the generated pdf to see the result.

Header / Pin	Symbol	Type	Description
Header1 - 1	GND	Power	Module / System GND
Header1 - 2	I03	I/O	GPIO – Capabilities are Module Dependent
Header1 - 3	I02	I/O	GPIO – Capabilities are Module Dependent
Header1 - 4	I01	I/O	GPIO – Capabilities are Module Dependent
Header1 - 5	3V3 OUT	Power	3.3V Power Output for User
Header2 - 1	RESET	I	System Reset, Active Low

In the example above, the [example section](#) is wrapped inside a `div` like below:

```
<div markdown class="rotated-page">

#### Example

In this example, we are going to change the page orientation for \[this subsection\](#example).

!!! note

    Download the generated pdf to see the result.

| Header / Pin | Symbol | Type | Description |
|:-----|:-----|:-----|:-----|
| Header1 - 1 | GND | Power | Module / System GND |
| Header1 - 2 | IO3 | I/O | GPIO – Capabilities are Module Dependent |
| Header1 - 3 | IO2 | I/O | GPIO – Capabilities are Module Dependent |
| Header1 - 4 | IO1 | I/O | GPIO – Capabilities are Module Dependent |
| Header1 - 5 | 3V3 OUT | Power | 3.3V Power Output for User |
| Header2 - 1 | RESET | I | System Reset, Active Low |
| Header2 - 2 | GND | Power | Module / System GND |
| Header2 - 3 | RX | I | Asynchronous Serial UART Receive Pin (TX from Host) |
| Header2 - 4 | TX | O | Asynchronous Serial UART Transmit Pin (RX from Host) |
| Header2 - 5 | 5V | Power | Module 5V Input, Main Power |

</div>
```

and the `custom.css` file contains this code:

```
:root {
  --base-page-orientation: a4 portrait;
  --rotated-page-orientation: a4 landscape;
}
```

#### Note

You can write your own custom CSS to handle page orientation but you must use the **named page** approach like below:

```
/* Named page ↓ */
@page rotated {
  size: A3 landscape;
}

.rotated-page {
  page: rotated;
  page-break-before: always;
  page-break-after: always;
}
```