

Customisation

Documentation

Duodu Randy



Table of Contents

1 Custom cover page	3
1.1 Using custom cover template	3
2 Adjusting the output	4

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

In the directory you set as `custom_template_path`, create a template file which the name `cover`. E.g. `cover.html` or `cover.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 which you can use in your `custom.css` file:

- `--title`
- `--subtitle`
- `--author`
- `--author-logo`
- `--copyright`
- `--type`
- `--site_url`
- `--revision`
- `--filename`
- `--chapter` (i.e. the H1 element in body content)

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

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.