Basildon

April 13, 2024

Contents

1 A static website generator

Basildon is a simple static website generator written in PHP and supporting Markdown content, Twig templates, SQLite, and outputs of HTML and PDF (via LaTeX).

This documentation is also available as a PDF[1].

1.1 Quick start

Prerequisites: PHP[2] (version 7.3 or higher) and Composer[3].

1. `composer create-project samwilson/basildon-skeleton mysite`

[1] `basildon-docs.pdf`
[2] `https://www.php.net/`
[3] `https://getcomposer.org/`

2. `cd mysite`
3. `./vendor/bin/basildon build .`
4. Edit files in the `content/` and `templates/` directories (for more details, see below).

1.2 Content

Content goes in the `content/` directory, in whatever structure is required. Each file comprises two parts:

- a frontmatter block of Yaml-formatted metadata; and
- and a text body after the frontmatter, in any format (the file's extension should match this, e.g. the default `.md` for Markdown).

Read more about Content.[4]

1.3 Templates

Templates are written in the Twig[5] language, and can output to any format required. Usually HTML is the target format, but LaTeX, XML, or anything else is just as possible. Formats do have to have a file extension though (that's how they're identified, in Basildon).

All templates live in the `templates/` directory of a site. The structure within that directory can be anything.

Read more about Templates.[6]

1.4 Assets (stylesheets and scripts)

Every stylesheet and script in the `assets/` directory will be copied to `output/assets/`.

Images should be in the `content/` directory; for more information, see the Content documentation page[7].

1.5 Output

All output is in the `output/` directory of a site. This directory is ready to be uploaded to a web server as the top level of the site.

The `output/` directory is emptied on every run of Basildon. However, sometimes you need to be able to keep files or directories that persist. For example, you might want `output/` to be its own Git repository for Github Pages, or to add a `_redirects` file for Netlify, or any number of other things. This is possible with the `output_exclude` config key, which takes an array of regular expressions to be matched against relative paths (these paths include the leading slash, similar to page IDs). For example:

[4] `content.html`
[5] `https://twig.symfony.com/`
[6] `templates.html`
[7] `content.html`

```
output_exclude:
  - "|/_redirects|"
  - "|/\\.git.*|"
```

2 Content

This page details the `content/` directory of a Basildon site. All pages of a site live in the content directory: each is a separate text file, and the file names and directory hierarchy are not prescribed. Content files have two parts: one, a Yaml-formatted frontmatter, delimited by three hyphens; and two, a main body that can be in any format. The file extension should match the format of the body; often this is Markdown (`.md`), but it doesn't have to be — you could easily have all your content files be HTML if that suits your site better.

An example of a page file at `content/topics/goats.md`:

```
---
title: Goats
description: Goats are good example animals.
---

This is the part where we explain more about the 'goat' topic.
```

Because content is usually in Markdown format[8], there are some useful Markdown additions that can be used in content pages. The rest of this page explains these.

All the metadata from content files is read into an SQLite[9] database when a site is built, which can be queried in templates[10] (see that page for more information about how). The database can also be modified and the changes written back to the content files[11].

2.1 Images

Local images should be stored in the `content/` directory, and included with the normal Markdown syntax. Their file paths should be relative to their own location and not start with a slash.

For example, an image file stored at `content/images/file.png` should be referenced like this:

- From `lorem.md` as `![Alt text](images/file.png)`.
- From `lorem/ipsum.md` as `![Alt text](../images/file.png)`.

For information about other assets such as stylesheets and scripts, see the *Assets* section[12] of the documentation overview.

[8] `https://www.markdownguide.org/getting-started/`
[9] `https://www.sqlite.org`
[10] `templates.html`
[11] `writing.html`
[12] `index.html`

2.2 Shortcodes

This section documents 'shortcodes', which are what we call specific replacable parts in a Markdown document. They are inline phrases or blocks of text such as `{foo}` or `{{{bar|id=123}}}` which get replaced by the contents of templates such as `templates/shortcodes/foo.html.twig` or `templates/shortcodes/bar.tex.twig`.

- Inline shortcodes are delimited by single braces and can contain any number of attributes, e.g.:
  - Lorem `{foo}` ipsum with no parameters.
  - Lorem `{foo | bar=baz|bif="foo bar"}` ipsum with two parameters, the second of which contains a space.
  - Lorem `{foo bif}` ipsum with a parameter with no value.
- Block shortcodes are delimited by triple braces at the beggining of lines, e.g.:
  - A block of one line, with one parameter:

    ```
    {{{quotation | cite="Author name"
    Lorem ipsum
    }}}
    ```

  - A single-line block with one parameter:

    ```
    {{{linebreak num=10}}}
    ```

Shortcodes replace an earlier feature in Basildon called 'embeds'. The functionality of embeds can be achieved with shortcodes, along with a lot more.

The term 'shortcode' (as well as the older 'embed') comes from WordPress, which has a similar function[13].

Shortcodes are a simple way to include images, videos, and summaries of other web pages. For example, this is a photo from Wikimedia Commons:

[13] `https://codex.wordpress.org/shortcode`

Figure 1: The Co-op, Post Office, and Courthouse on Stirling Terrace in York, Western Australia.

It is added to the source Markdown with this:

```
{{{commons|Co-Op,_Post_Office,_Courthouse.jpg}}}
```

All of the other information (image URL, caption, etc.) is retrieved from the Commons API when the Markdown is rendered.

Shortcodes can be rendered to any output format; they're not limited to HTML.

2.2.1 Configuration

To configure a new shortcode, add a file to the templates' directory, with a name matching what you want to use in the Markdown.

The file `templates/shortcodes/<shortcode-name>.<format>.twig` to contain the HTML or other output that should be output for the shortcode.

The following variables are available in shortcode templates:

- `shortcode.name`: the name of the shortcode, which will always be the same as the template's name.
- `shortcode.attrs.foo`: fetches an attribute by name.
- `shortcode.attrs.1`: fetches an unnamed attribute by number (starting from 1).
- `shortcode.body`: for block shortcodes, fetches the entire body text.

2.2.2 Example: Wikimedia Commons

In any Markdown file:

```
{{{commons file=Example.jpg}}}
```

In `templates/shortcodes/commons.html.twig`:

```
{% set commons = commons(shortcode.attr('file')) %}
<figure>
    <a href="{{ commons.imageinfo.0.descriptionurl }}">
        <img src="{{ image_url( commons.imageinfo.0.thumburl ) }}"
             width="{{ commons.imageinfo.0.thumbwidth }}"
             height="{{ commons.imageinfo.0.thumbheight }}"
             alt="{{ commons.labels.en.value|escape('html_attr') }}"
        />
    </a>
    <figcaption>{{ commons.labels.en.value }}</figcaption>
</figure>
```

Note that this is also using the `commons()` Twig function, which is documented separately[14].

2.2.3 Example: Flickr

In any Markdown file:

```
{{{flickr|id=123456}}}
```

In `templates/shortcodes/flickr.html.twig`:

```
{% set flickr = flickr(shortcode.attrs.id) %}

<figure itemscope itemtype="http://schema.org/ImageObject">
    <a href="{{ flickr.urls.photopage }}"><img alt="An image from Flickr."
    <figcaption>
        <strong itemprop="name">{{ flickr.title }}{% if flickr.description
        {% if flickr.description %}
            <span itemprop="description">{{ flickr.description|raw }}</span
        {% endif %}
        <span class="meta">
            {% if flickr.dates.taken %}
                {% set date = date_create(flickr.dates.taken) %}
                <time datetime="{{ date.format('c') }}">{{ date.format('Y F
            {% endif %}
            &middot; <a href="{{ flickr.urls.photopage }}">via Flickr</a>
            &middot; <a href="{{ flickr.license.url }}" rel="license" title
        </span>
    </figcaption>
</figure>
```

3 Templates

Templates in Basildon are all written in the Twig[15] templating language. They can output any format that's required. Basildon provides a few variables func-

[14] `./templates.html`
[15] `https://twig.symfony.com/`

tions, and filters for common website use cases; these are explained on this page.

3.1 Variables

1. `page` – An object representing the current page being rendered. It has the following members:
   - `page.body`: The unmodified body text, good for piping through filters such as `md2html` and `md2latex`.
   - `page.metadata`: All the metadata defined in the page's frontmatter.
   - `page.link(target)`: Creates a relative URL to another page.
2. `database` - The database, the most useful attribute of which is `database.query(sql)`.
3. `site` – The site object, mostly used to access configuration values, e.g. `site.config.title`.

3.2 Functions

1. `commons(file_name)` – Get information about a Wikimedia Commons[16] file.
2. `flickr(photo_id)` – Get information about a Flickr[17] photo. To use this, you need to set the `flickr.api_key` and `flickr.api_secret` values in your site's `config.local.yaml` file.
3. `qrcode(text)` – Returns an asset-directory path to a QR code SVG file, such as `/assets/8a482ae2afb51a1de85b7eb9087f7cc2.svg`. For example: `<img src="{{ page.link(qrcode('string')) }}" />`
4. `wikidata(qid)` – Returns information about the given Wikidata[18] item. For example, `{{ wikidata('Q42').descriptions.en.value }}` will return something like "English writer and humorist". To get the full details of the returned structure, see e.g. wikidata.org/wiki/Special:EntityData/Q42.json[19].
5. `wikidata_query(sparql)` — Returns the result of the Sparql query from Wikidata. See the example in /example/templates/tag.html.twig[20].
6. `commons_query(sparql)` — Returns the result of a Sparql query on Wikimedia Commons. This requires an authentication token to be added to `config.yaml`. Instructions for retrieving this token can be found on Commons[21], and an example for how to use the function is in /example/templates/shortcodes/commons_depicts_count.html.twig[22].

[16] `https://commons.wikimedia.org/`
[17] `https://www.flickr.com/`
[18] `https://www.wikidata.org/`
[19] `https://www.wikidata.org/wiki/Special:EntityData/Q42.json`
[20] `https://github.com/samwilson/basildon/blob/main/example/templates/tag.html.twig`
[21] `https://commons.wikimedia.org/wiki/Commons:SPARQL\_query\_service/API\_endpoint`
[22] `https://github.com/samwilson/basildon/blob/main/example/templates/shortcodes/commons\_depicts\_count.html.twig`

7. `wikipedia(lang, title)` — Returns an HTML extract of the given article. For example: `{{wikipedia('en', 'Tag (metadata)')|raw}}`

8. `get_json(url)` — Fetch JSON data from any URL. For example: `{{get_json('https://api.wikitre 22337').0.profile.LongName}}`

9. `get_feeds(urls)` — Fetch RSS or Atom feed items. The `urls` parameter can be a single URL string or an array, and the URLs can be of the feed or the website for which to attempt autodiscovery. An array is returned, each element of which is a Simplepie Item[23]. For example: `{{get_json('https://samwilson.id.au/news.rss')}}`

3.3 Filters and escapers

1. `md2html` – Filter markdown to HTML.

2. `md2latex` – Filter markdown to LaTeX.

3. `escape('tex')` – Escaper to use in TeX templates to escape characters that have special meaning in TeX, e.g. `{{ '$10'|e('tex') }}`. This is often used by wrapping the template in `{% autoescape 'tex' %}{% endautoescape %}`

4. `dirname` and `basename` — Identical to PHP's dirname()[24] and basename()[25] functions. Useful for working with Basildon page IDs.

4 Writing

Basildon supports bulk writing of metadata into Markdown frontmatter.

All the metadata from content files is read into an SQLite[26] database when a site is built, and this database can be opened in any other programme and edited. After being saved to the database, the `build` command can be run to update the `content/` directory files.

The whole workflow should be something like:

1. Make sure your site is under version control, and has no outstanding changes (to make it easier to track changes that will be made by Basildon).

2. Build your site: `./vendor/bin/basildon build .`

3. Edit the database at `./cache/database/db.sqlite3` using a programme such as DB Browser for SQLite[27], and write the changes back to the same file.

4. Run `./vendor/bin/basildon write .`

5. Check the changes before committing them.

[23] `https://github.com/simplepie/simplepie/blob/1.8.0/src/Item.php`
[24] `https://www.php.net/manual/en/function.dirname.php`
[25] `https://www.php.net/manual/en/function.basename.php`
[26] `https://www.sqlite.org`
[27] `https://sqlitebrowser.org/`