Exhibition Documentation

Release 0.0.1

Matt Molyneaux

Jul 04, 2018

Contents:

Say it right:

> /gs’hb’()n/

So something like:

> eggs hib ish’n

CHAPTER 1

What?

A static site generator

CHAPTER 2

License?

GPLv3 or later. See LICENSE for the actual text.

CHAPTER 3

Why though?

I've been using Hyde since forever, but I wasn't happy with it. I was also very unhappy with other static site generators (SSGs) that used Jinja2 for their templating needs:

- Pelican and the like are too blog focused. It didn't feel in the spirit of those projects to have a blog and a recipe list as two separate sections to a website.
- Hyde is everything I want, except for the complete lack of documentation and a massive code base that needs a lot of work to make it run on Python 3. It is also currently unmaintained.
  - I should also mention that there are huge parts of Hyde that do nothing for me, so starting from scratch made more sense than dealing with Hyde.

There are SSGs that aren't written in Python or don't use Jinja2 for their templates, but I'm not interested in rewritting all the templates for the sites that I have made with Hyde.

CHAPTER 4

What's the status of this project?

I'm not using it for anything serious yet, but there are tests, and there are some docs.

4.1 Getting started

Exhibition is fairly quick to configure.

4.1.1 Minimum setup

At minimum, Exhibition expects to find a YAML file, `site.yaml`, with at least `deploy_path` and `content_path` defined. The path specified in `content_path` needs to exist.

For example:

```
$ mkdir content
$ cat << EOF > site.yaml
> deploy_path: deploy
> content_path: content
> EOF
```

You can now generate your first Exhibition website!:

```
$ exhibit gen
$ ls deploy
```

Of course, you've got no content so the directory will be empty.

Any file or directory you put in `content` will appear in `deploy` when you run `exhibit gen`.

4.1.2 Templates

Exhibition supports Jinja2 out of the box, but it needs to be enabled:

Listing 1: site.yaml

```
deploy_path: deploy
content_path: content
filter: exhibition.filters.jinja2
```

Now we can create HTML files that use Jinja2 template syntax:

Listing 2: content/index.html

```
<html>
  <body>
    <p>This page has {{ node.siblings|length }} siblings</p>
  </body>
</html>
```

Note: `node` is the current page being rendered and is passed to Jinja2 as a context variable.

Run `exhibit gen` and then `exhibit serve`. If you connect to `http://localhost:8000` you'll see the following text:

```
This page has 0 siblings
```

If you add another page, this number will increase when run `exhibit gen` again.

If you wish to use template inheritance, add the following to `site.yaml`:

```
templates: mytemplates
```

Where "mytemplates" is whatever directory you will store your templates in. You can either use the extends tag directly or you can specify `extends` in `site.yaml`. You can also specify `default-block` to save you from wrapping every page in `{% block content %}`:

```
extends: page.j2
default-block: content
```

And then our template:

Listing 3: mytemplates/page.j2

```
<html>
  <body>
    {% block content %}{% endblock %}
  </body>
</html>
```

Our index page would be this:

Listing 4: content/index.html

```
<p>This page has {{ node.siblings|length }} siblings</p>
```

The generated HTML will be exactly the same, except now files in `content/` will not have to each have their own copy of any headings, page title, links to CSS or whatever.

4.1.3 Meta

Site settings are available in templates as `node.meta`. For example:

Listing 5: content/otherpage.html

```
<p>Current filter is "{{ node.meta.filter }}"</p>
```

Which will generate the following:

```
Current filter is "exhibition.filters.jinja2"
```

You can reference any data that you put in `site.yaml` like this - and there's no limit on what you can put in there.

As well as `site.yaml` there are two additional places that settings can be controlled: `meta.yaml` and front matter.

Meta files

A `meta.yaml` can be used to define or override settings for a particular directory and any files or subdirectories it contains.

Let's add a blog to our website:

```
$ mkdir content/blog
$ cat << EOF > content/blog/meta.yaml
> extends: blog_post.j2
```

Now all HTML files in `content/blog/` will use the `blog_post.j2` as their base template rather than `page.j2`, but files such as `content/index.html` will still use `page.j2` as their base template.

Note: `meta.yaml` files do not appear as nodes and won't appear in `deploy_path`

Front matter

Front matter is the term used to describe YAML metadata put at the beginning of a file. Unlike `meta.yaml`, any settings defined (or overridden) here will only affect this one file.

For example, we won't want the index page of our blog to use `blog_post.j2` as its base template:

Listing 6: content/blog/index.html

```
---
extends: blog_index.j2
---
{% for post in node.sibling %}
   <p><a href="{{ post.full_url }}">{{ post.meta.title }}</a></p>
```

Listing 7: content/blog/first-post.html

```
---
title: My First Post
---
<h1>{{ node.meta.title }}
<p>Hey! This is my first blog post!</p>
```

4.1.4 What next?

Checkout the *API*. File bugs. Submit patches.

Exhibition is still in the early stages of development, so please contribute!

4.2 exhibit commandline script

4.2.1 exhibit

```
exhibit [OPTIONS] COMMAND [ARGS]...
```

Options

-v, --verbose
Verbose output, can be used multiple times to increase logging level

gen

Generate site from content_path

```
exhibit gen [OPTIONS]
```

serve

Serve files from deploy_path as a webserver would

```
exhibit serve [OPTIONS]
```

4.3 exhibition

4.3.1 exhibition package

Subpackages

exhibition.filters package

Submodules

exhibition.filters.jinja2 module

Jinja2 template filter

To use, add the following to your configuration file:

```
filter: exhibition.filters.jinja2
```

class exhibition.filters.jinja2.**Mark**(*environment*)

Bases: `jinja2.ext.Extension`

Marks a section for use later:

```
{% mark intro %}
<p>My Intro</p>
{% endmark %}

<p>Some more text</p>
```

This can then be referenced via `Node.marks`.

identifier = 'exhibition.filters.jinja2.Mark'

parse(*parser*)

If any of the *tags* matched this method is called with the parser as first argument. The token the parser stream is pointing at is the name token that matched. This method has to return one or a list of multiple nodes.

tags = {'mark'}

class exhibition.filters.jinja2.**RaiseError**(*environment*)

Bases: `jinja2.ext.Extension`

Raise an exception during template rendering:

```
{% raise "This is an error" %}
```

identifier = 'exhibition.filters.jinja2.RaiseError'

parse(*parser*)

If any of the *tags* matched this method is called with the parser as first argument. The token the parser stream is pointing at is the name token that matched. This method has to return one or a list of multiple nodes.

tags = {'raise'}

exhibition.filters.jinja2.**content_filter**(*node*, *content*)

This is the actual content filter called by *exhibition.main.Node* on appropiate nodes.

Parameters

- **node** – The node being rendered
- **content** – The content of the node, stripped of any YAML front matter

exhibition.filters.jinja2.**markdown**(*ctx*, *text*)

exhibition.filters.jinja2.**metareject**(*nodes*, *key*)

exhibition.filters.jinja2.**metaselect**(*nodes*, *key*)

exhibition.filters.jinja2.**metasort**(*nodes*, *key=None*, *reverse=False*)

Sorts a list of nodes based on keys found in their meta objects

Submodules

exhibition.command module

Documentation for this module can be found in *exhibit commandline script*

exhibition.main module

class `exhibition.main.`**`Config`**(*data=None*, *parent=None*)

Bases: `object`

Configuration object that implements a dict-like interface

If a key cannot be found in this instance, the parent *`Config`* will be searched (and its parent, etc.)

Parameters

- **`data`** – Can be one of a string, a file-like object, a dict-like object, or `None`. The first two will be assumed as YAML
- **`parent`** – Parent *`Config`* or `None` if this is the root configuration object

`copy`()

`classmethod from_path`(*path*)

Load YAML data from a file

`get`(*key*, *default=None*)

`items`()

`keys`()

`load`(*data*)

Load data into configutation object

Parameters `data` – If a string or file-like object, `data` is parsed as if it were YAML data. If a dict-like object, `data` is added to the internal dictionary.

Otherwise an `AssertionError` exception is raised

`update`(**args*, ***kwargs*)

`values`()

class `exhibition.main.`**`Node`**(*path*, *parent*, *meta=None*)

Bases: `object`

A node represents a file or directory

Parameters

- **`path`** – A `pathlib.Path` that is either the `content_path` or a child of it.
- **`parent`** – Either another *`Node`* or `None`
- **`meta`** – A dict-like object that will be passed to a *`Config`* instance

`add_child`(*child*)

Add a child to the current Node

If the child doesn't already have its `parent` set to this Node, then an `AssertionError` is raised.

data
Extracts data from contents of file

For example, a YAML file

classmethod from_path(*path*, *parent=None*, *meta=None*)
Given a `pathlib.Path`, create a Node from that path as well as any children

If the path is not a file or a dir, an `AssertionError` is raised

Parameters

- **path** – A `pathlib.Path` that is either the `content_path` or a child of it.
- **parent** – Either another *`Node`* or `None`
- **meta** – A dict-like object that will be passed to a *`Config`* instance

full_path
Full path of node when deployed

full_url
Get full URL for node, including trailing slash

get_content()
Get the actual content of the Node

First calls *`process_meta()`* to find the end any front matter that might be present and then returns the rest of the file

If `filter` has been specified in *`meta`*, that filter will be used to further process the content.

marks
Marked sections from content

Calls *`get_content()`* to process content if that hasn't been done already

meta
Configuration object

Automatically loads front-matter if applicable

process_meta()
Finds and processes the YAML fonrt matter at the top of a file

If the file does not start with `---\n`, then it's assumed the file does not contain any meta YAML for us to process

render()
Process node and either create the directory or write contents of file to `deploy_path`

siblings
Returns all children of the parent Node, except for itself

walk(*include_self=False*)
Walk through Node tree

`exhibition.main.`**gen**(*settings*)
Generate site

Deletes `deploy_path` first.

`exhibition.main.`**serve**(*settings*)
Serves the generated site from `deploy_path`

Respects settings like `base_url` if present.

4.4 Changelog

4.4.1 0.0.1

Everything is new! Some choice features:

- Configuration via YAML files and YAML front matter
- Jinja2 template engine is provided by default
- A local HTTP server for development work
- Less than 2000 lines of code, including tests

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index