Exhibition Documentation

Release 0.0.4

Matt Molyneaux

Aug 27, 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 rewriting all the templates for the sites that I have made with Hyde.

CHAPTER 4

What's the status of this project?

There are tests, there's some documentation, and I currently use it for a number of websites including my personal blog.

Please feel free to add your site to the wiki if it uses Exhibition, but please make sure its safe for work and not covered in adverts.

4.1 Contributions

I'm always looking for contributions, whether they be bug reports, bug fixes, feature requests, new features, or documentation. Also, feel free to open issues for support requests too - these are very helpful in showing me where documentation is required or needs improving.

There are however some items I won't consider for inclusion:

- Functionality to upload the static site once generated. This is and shall remain out of scope for this project.
- Windows support. I've tried maintaining packages before that have Windows support. I usually end up breaking it as I have no way to test out my changes on a regular basis.
- Python 2 support.

4.1.1 Getting started

Exhibition is fairly quick to configure.

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`.

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.

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 frontmatter.

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`

Frontmatter

Frontmatter 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>
```

What next?

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

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

4.1.2 exhibit commandline script

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]
```

4.1.3 Configuration

Exhibition draws configuration options from three places:

- `site.yaml`, which is the root configuration file
- `meta.yaml`, which there can be one or none in any given folder
- "Frontmatter", which is a YAML header that can be put in any text file. It *must* be the first thing in the file and it *must* start and end with `---` - both on their own lines.

The difference between these different places to put configuration is explain in detail in the *Getting started* page.

Inheritance

One important aspect of Exhibition's configuration system is that for a given node (a file or a folder), a key is search for in the following way:

1. The current node is checked for the specified key. If it's found, it is returned. Otherwise carry on to 2.
2. The parent of the current node is checked, and if the specified key is not found here then *its* parent is checked the same way (and so on), until the root node is found.
3. If the root node does not have the specified key, then `site.yaml` is searched.
4. Only once `site.yaml` has been searched is a `KeyError` raised.

Mandatory

The following options must be present in `site.yaml`:

`content_path`

This is the path to where Exhibition will load data from. It should have the same directory structure and files as you want to appear in the rendered site.

`deploy_path`

Once rendered, pages will be placed here.

Warning: `content_path` and `deploy_path` should *only* appear in `site.yaml`.

General

`ignore`

Matching files are not processed by Exhibition at all. Can be a file name or a glob pattern:

```
ignore: "*.py"
```

As glob patterns are fairly simple, `ignore` can also be a list of patterns:

```
ignore:
  - "*.py"
  - example.xcf
```

`base_url`

If your site isn't deployed to the root of a domain, use this setting to tell Exhibition about the prefix so it can be added to all URLs

Filters

‘ *filter*‘

The dotted path notation that Exhibition can import to process content on a node.

Exhibition comes with one filter: `exhibition.filters.jinja2`

`filter_glob`

Matching files are processed by `filter` if specified, otherwise this option does nothing.

```
filter_glob: "*.html"
```

As glob patterns are fairly simple, `filter_glob` can also be a list of patterns:

```
filter_glob:
  - "*.html"
  - "*.htm"
  - "robot.txt"
```

Filters specify their own default glob, refer to the documentation of that filter to find out what that is.

Jinja2

`templates`

The path where Jinja2 templates will be found. Can be single string or a list.

`extends`

If specified, this will insert a `{% extends %}` statement at the beginning of the file content before it is passed to Jinja2.

`default_block`

If specified, this will wrap the file content in `{% block %}`.

`markdown_config`

Markdown options as specified in the Markdown documentation.

Cache busting

Cache busting is an important tool that allows static assets (such as CSS files) to bypass the browser cache when the content of such files is updated, while still allowing high value expiry times.

`cache_bust_glob`

Matching files have their deployed path and URL changed to include a hash of their contents. E.g. `media/site.css` might become `media/site.894a4cd1.css`. You can specify globs in the usual manner:

```
cache_bust_glob: "*.css"
```

As glob patterns are fairly simple, `cache_bust_glob` can also be a list of patterns:

```
cache_bust_glob:
  - "*.css"
  - "*.jpg"
  - "*.jpeg"
```

To refer to cache busted nodes in your Jinja2 templates, do the following:

```
<link rel="stylesheet" href="{{ node.get_from_path("/media/css/site.css").full_url }}
↪" type="text/css">
```

4.1.4 exhibition

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 frontmatter

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*, *node=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
- **`node`** – The node that this object to bound to, or `None` if it is the root configuration object

`copy`()

`classmethod from_path`(*path*)

Load YAML data from a file

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

`get_name`()

`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.

`cache_bust`

`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 frontmatter 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.

get_from_path(*path*)

Given a relative or absolute path, return the *`Node`* that represents that path.

Parameters path – A `str` or `pathlib.Path`

marks

Marked sections from content

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

meta

Configuration object

Automatically loads frontmatter 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.

4.1.5 Changelog

0.0.4

- Added vesrioneer
- Fix bug where `exhibit serve` was not serving files with extension stripping enabled
- `KeyError``s raised by ``Config` now display the path of the node they are attached to, making debuging missing keys far easier.
- Improved test coverage and fixed numerous bugs
- Implemented cache busting for static assets (images, CSS, and such). Use the `cache_bust_glob` option to control which files are cache busted.
- Implemented `Node.get_from_path` which can fetch a *`exhibition.main.Node`* specified by a path
- Make all Exhibition defined meta keys use underscores not hyphens

0.0.3

- Fix bug where extension stripping was not being applied

0.0.2

- Fixed trove classifiers
- Add `__version__` to `exhibition.__init__`

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