Exhibition Documentation

Release 0.2.1

Matt Molyneaux

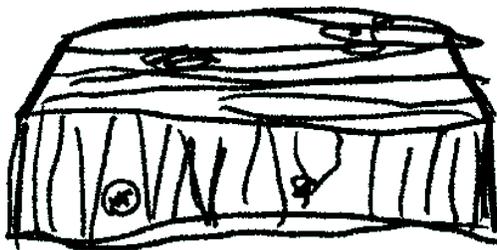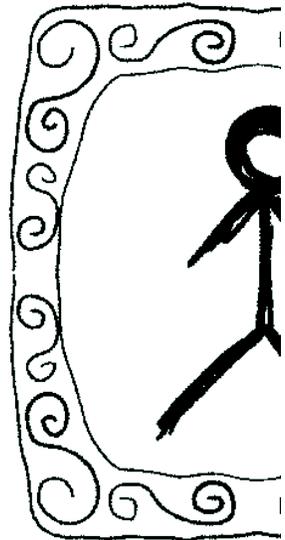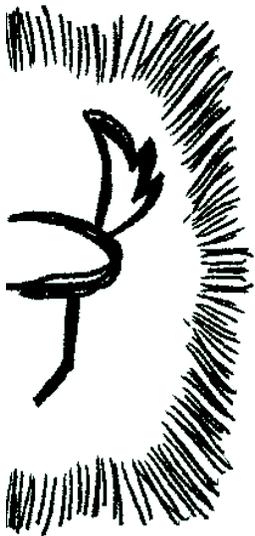**May 01, 2021**

CONTENTS:

Say it right:

/gs'hb'()n/

So something like:

eggs hib ish'n

WHAT?

A static site generator

WHERE?

- Download: https://pypi.org/project/exhibition/
- Source: https://github.com/moggers87/exhibition
- Docs: https://exhibition-ssg.readthedocs.io/en/latest/

LICENSE?

GPLv3 or later. See LICENSE for the actual text.

FOUR

WHY THOUGH?

I'd been using Hyde for a number of years, eventually that project stopped receiving updates. Hyde had very limited test coverage, many features that I didn't personally use, and no Python 3 support. This combination made the prospect of maintaining Hyde daunting, so forking was out of the question.

In the end, I wrote Exhibition as other available static site generators would either require massively rewriting the sites I already had or weren't flexible enough to generate the same URL structure.

FIVE

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.

5.1 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.

SIX

CODE OF CONDUCT

The Exhibition project has adopted the Contributor Covenant Code version 1.4. By contributing to this project, you agree to abide by its terms.

The full text of the code of conduct can be found here

6.1 Getting started

Exhibition is fairly quick to configure.

6.1.1 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
```

Note: Exhibition comes with a starter template for new sites. See `exhibit create --help` for more information.

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`.

6.1.2 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.

6.1.3 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 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>
```

6.1.4 What next?

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

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

6.2 `exhibit` commandline script

6.2.1 exhibit

A Python static site generator

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

Options

`--version`
    Show the version and exit.

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

create

Generate a starter site with a basic configuration for website.

```
exhibit create [OPTIONS] PATH
```

Options

`-f, --force`
    overwrite destination if it exists

Arguments

PATH
    Required argument

gen

Generate site from content_path

```
exhibit gen [OPTIONS]
```

serve

Serve files from deploy_path as a webserver would

```
exhibit serve [OPTIONS]
```

Options

-s, --server `<server>`
    Hostname to serve the site at.

-p, --port `<port>`
    Port to serve the site at.

6.3 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.

6.3.1 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 if the key cannot be found.

6.3.2 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`.

6.3.3 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

strip_exts

Specifies if certain files should have their extensions removed when being referenced via `Node.full_url`. When deploying, this does not change the filename, so you will need to configure your web server to serve those files correctly.

By default, this will be applied to files ending in `.html`, to disable this feature use the following:

```
strip_exts:
```

You can also specify multiple file extensions:

```
strip_exts:
    - .html
    - .htm
```

index_file

Specify the name file name of the "index" file for a directory. By default this is `index.html`, as it is on most web servers. If you change this settings, be sure to update your web server's configuration to reflect this change.

6.3.4 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.

External Command

`external_cmd`

The command to run. This should use the placeholders `{INPUT}` and `{OUTPUT}` for the input and output files respectively. For example:

```
external_cmd: "cat {INPUT} | sort > {OUTPUT}"
```

6.3.5 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">
```

6.4 Filters

Exhibition comes with a number of filters. You can also write your own!

6.4.1 Jinja2

`exhibition.filters.jinja2` will process the contents of a node via the [Jinja2 templating engine](). Check the Jinja2 documentation for syntax and a basic understanding of how Jinja2 works.

Unless it is set, `filter_glob` will default to `*.html`

Context variables

> **node** The current node
>
> **time_now** A datetime object that contains the current time in UTC.

Meta

There are some meta options that are used exclusively by Jinja2:

> **templates** Where to search for templates.
>
> **extends** Automatically add an `{% extends %}` tag to the start of the content of every affected node.
>
> **default_block** Wrap the content of affected nodes with the specificity `{% block %}` tag.

Markdown

`markdown` is provided as a filter and it can be configured via the `markdown_config` meta variable, which is passed to the markdown function as keyword arguments.

Please view the [Markdown documentation]() for details.

Pandoc

`pandoc` is provided as a filter and can be configured by the `pandoc_config` meta variable, which is passed to the convert_text function as keyword arguments.

Please refer [pypandoc project]() for details.

Note, pypandoc requires pandoc to be installed. It will error without it.

Typogrify

All Typogrify filters are available. See the [Typogrify webste]() for more details.

Exhibition specific filters

`metasort`

Given a list of nodes, `metasort` will sort the list like this:

```
{{ node.children.values()|metasort("somekey") }}
```

Where `somekey` is a key found in each node's meta.

You can also reverse the order like so:

```
{{ node.children.values()|metasort("somekey", True) }}
```

metaselect and metareject

Given a list of nodes, `metaselect` will filter out nodes that either do not have that key in their meta or do but the value resolves to something falsey. For example, the following will filter out any nodes that have `listable` set to `False`:

```
{{ node.children.values()|metaselect("listable") }}
```

`metareject` works the same way, except it filters out nodes that *don't* have falsey values for the given key.

Marked sections

Marked sections are a great way to allow parts of your content to be referenced elsewhere, for example the preamble to a blog post:

```
---
title: My Post
---
{% mark intro %}
Blah blah blah...
{% endmark %}

Some more text
```

In another node you might want to list all the blog posts with their intros:

```
{% for child in node.children.values() %}
    <h3>{{ node.meta.title }}</h3>
    <p>{{ node.marks.intro }}</p>
{% endfor %}
```

You can have as many marks as you like in a node and they can be nested.

Raising Errors

Sometimes it can be useful to raise an error, especially if the logic in your template is quite complex!

```
{% if 2 == 3 %}
    {% raise "This shouldn't be true! The Universe is broken!" %}
{% endif %}
```

6.4.2 External Command

The external command filter only has one option: `external_cmd`, which is the shell command to be run. The specified command should use `{INPUT}` as the input file and `{OUTPUT}` as the output file, for example:

```
external_cmd: "cat {INPUT} | base64 > {OUTPUT}"
```

Unless it is set, `filter_glob` will default to `*.*` for this filter.

6.4.3 Make Your Own

To create your own filter for Exhibition, your module must implement a function with the following signature:

```
def content_filter(node, content):
    return ""
```

> **node** is the current node being processed.
>
> **content** is the content of that node, with any frontmatter removed.

`content_filter` should return a string, which will then become the rendered form of this node.

6.5 exhibition

6.5.1 exhibition package

Subpackages

exhibition.filters package

Submodules

exhibition.filters.external module

External command filter

Use an external command to process a file, like so:

```
filter: exhibition.filters.external
external_cmd: sed 's/this/that/g' {INPUT} > {OUTPUT}
```

exhibition.filters.external.**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 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'}
       if this extension parses this is the list of tags it's listening to.

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'}
       if this extension parses this is the list of tags it's listening to.

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

exhibition.filters.jinja2.**pandoc**(*ctx*, *text*, *fmt=None*)

Module contents

Submodules

exhibition.command module

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

exhibition.config module

class exhibition.config.**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()

exhibition.node module

class `exhibition.node.`**`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.
>
> **property `cache_bust`**
>
> **property `content`**
>
>> Get the actual content of the Node
>>
>> If `filter` has been specified in *meta*, that filter will be used to further process the content.
>
> **property `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. Children are loaded in Unicode codepoint order - this order is preserved in `Node.children` if you're unsure what that means.
>>
>> 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
>
> **property `full_path`**
>
>> Full path of node when deployed
>
> **property `full_url`**
>
>> Get full URL for node, including trailing slash
>
> **`get_from_path`**(*path*)
>
>> Given a relative or absolute path, return the *Node* that represents that path.
>>
>>> **Parameters** **path** – A `str` or `pathlib.Path`
>
> **property `index_file`**
>
> **property `marks`**
>
>> Marked sections from content
>
> **property `meta`**
>
>> Configuration object
>>
>> Finds and processes the YAML front 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`

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

property strip_exts

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

exhibition.utils module

class exhibition.utils.**ExhibitionBaseHTTPRequestHandler**(**args*, *directory=None*, ***kwargs*)
    Bases: `http.server.SimpleHTTPRequestHandler`

    end_headers()
        Send the blank line ending the MIME headers.

    translate_path(*path*)
        Translate a */*-separated PATH to the local filename syntax.

        Components that mean special things to the local file system (e.g. drive or directory names) are ignored. (XXX They should probably be diagnosed.)

exhibition.utils.**gen**(*settings*)
    Generate site

    Deletes `deploy_path` first.

exhibition.utils.**serve**(*settings*, *server_address*)
    Serves the generated site from `deploy_path`

    Respects settings like `base_url` if present.

Module contents

6.6 Security

If you find a security issue with Exhibition, email security@moggers87.co.uk. If you want to send an encrypted report, then please use key id 0x878B5A2A1D47C084.

Exhibition follows the same security reporting model that has worked for other open source projects: If you report a security vulnerability, it will be acted on immediately and a fix with complete full disclosure will go out to everyone at the same time.

6.7 Changelog

6.7.1 0.2.1

This release is just for a fix some bad syntax in our documentation. Nothing else has changed!

0.2.0

6.7.2 Added

- Add support for a pandoc filter for Jinja2 templates
- `exhibit create` command to get users started

6.7.3 Removed

- Remove support for Python 3.5, 3.6, and 3.7

6.7.4 Fixed

- Add description to `exhibit` command
- Use `cached_property` to make code cleaner

0.1.1

6.7.5 Added

- Allow settings of local HTTP server address and port.
- Added Python 3.8 support.

6.7.6 Removed

Nothing was removed in this release

6.7.7 Fixed

- Set `Cache-Control` to `no-store` on HTTP server respones
- HTTP server should ignore `GET` params and fragments

0.1.0

The *"I'd almost recommand it to my friends"* release.

6.7.8 Added

- Added Python 3.7 support.

- Add the external command filter.

- Document Jinja2 filter.

- Add `strip_exts` as an user configurable setting.

- Add `index_file` as an user configurable setting.

6.7.9 Removed

- Removed Python 3.4 support.

6.7.10 Fixed

- Reorganised package so that code is easier to manage.

- Make node loading deterministic, meta files loaded first and then alphabetical order for the rest.

0.0.4

- Added vesrioneer.

- Fix bug where `exhibit serve` was not serving files with extension stripping enabled.

- A `KeyError` 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.

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

G

I

K

L

M

N

P

R

S

T

U

V

W