# staticjinja Documentation

*Release 0.4.0*

Ceasar Bautista

Nov 14, 2020

# Contents

staticjinja is a library for easily deploying static sites using the Jinja2 template engine.

Most static site generators are cumbersome to use. Nevertheless, when deploying a static website that could benefit from factored out data or modular HTML pages (especially convenient when prototyping), a templating engine can be invaluable. Jinja2 is an extremely powerful tool in this regard.

staticjinja is designed to be lightweight, easy-to-use, and highly extensible, enabling you to focus on simply making your site.

```
$ mkdir templates
$ vim templates/index.html
$ staticjinja watch
Building index.html...
Templates built.
Watching 'templates' for changes...
Press Ctrl+C to stop.
```

CHAPTER 1

# User Guide

This part of the documentation focuses on step-by-step instructions for getting the most of staticjinja.

## 1.1 Quickstart

Eager to get started? This page gives a good introduction for getting started with staticjinja.

### 1.1.1 Installation

staticjinja supports Python 2.6, 2.7, 3.3 and 3.4.

Installing staticjinja is simple with pip:

```
$ pip install staticjinja
```

### 1.1.2 Rendering templates

If you're just looking to render simple data-less templates, you can get up and running with the following shortcut:

```
$ staticjinja build
 Rendering index.html...
```

This will recursively search `./templates` for templates (any file whose name does not start with `.` or _) and build them to `.`.

To monitor your source directory for changes, and recompile files if they change, use `watch`:

```
$ staticjinja watch
 Rendering index.html...
 Watching 'templates' for changes...
 Press Ctrl+C to stop.
```

### 1.1.3 Configuration

`build` and `watch` each take 3 options:

- `--srcpath` - the directory to look in for templates (defaults to `./templates`);
- `--outpath` - the directory to place rendered files in (defaults to `.`);
- `--static` - the directory (or directories) within `srcpath` where static files (such as CSS and JavaScript) are stored. Static files are copied to the output directory without any template compilation, maintaining any directory structure. This defaults to `None`, meaning no files are considered to be static files. You can pass multiple directories separating them by commas: `--static="foo,bar/baz,lorem"`.

More advanced configuration can be done using the staticjinja API, see *Using Custom Build Scripts* for details.

## 1.2 Advanced Usage

This document covers some of staticjinja's more advanced features.

### 1.2.1 Partials and ignored files

A **partial file** is a file whose name begins with a _. Partial files are intended to be included in other files and are not rendered. If a partial file changes, it will trigger a rebuild if you are running `staticjinja watch`.

An **ignored file** is a file whose name begins with a `.`. Ignored files are neither rendered nor used in rendering templates.

If you want to configure what is considered a partial or ignored file, subclass `Site` and override `is_partial` or `is_ignored`.

### 1.2.2 Using Custom Build Scripts

The command line shortcut is convenient, but sometimes your project needs something different than the defaults. To change them, you can use a build script.

A minimal build script looks something like this:

```python
from staticjinja import Site


if __name__ == "__main__":
    site = Site.make_site()
    # enable automatic reloading
    site.render(use_reloader=True)
```

To change behavior, pass the appropriate keyword arguments to `Site.make_site`.

- To change which directory to search for templates, set `searchpath="searchpath_name"` (default is `./templates`).
- To change the output directory, pass in `outpath="output_dir"` (default is `.`).
- To add Jinja extensions, pass in `extensions=[extension1, extension2, ...]`.
- To change which files are considered templates, subclass the `Site` object and override `is_template`.

**Note:** Deprecated since version 0.3.4: Use `Make` or similar to copy static files. See Issue #58

To change where static files (such as CSS or JavaScript) are stored, set `staticpaths=["mystaticfiles/"]` (the default is `None`, which means no files are considered to be static files). You can pass multiple directories in the list: `staticpaths=["foo/", "bar/"]`. You can also specify singly files to be considered as static: `staticpaths=["favicon.ico"]`.

Finally, just save the script as `build.py` (or something similar) and run it with your Python interpreter.

```
$ python build.py
Building index.html...
Templates built.
Watching 'templates' for changes...
Press Ctrl+C to stop.
```

### 1.2.3 Loading data

Some applications render templates based on data sources (e.g. CSVs or JSON files).

The simplest way to supply data to templates is to pass `Site.make_site()` a mapping from variable names to their values (a "context") as the `env_globals` keyword argument.

```
if __name__ == "__main__":
    site = Site.make_site(env_globals={
        'greeting':'Hello world!',
    })
    site.render()
```

Anything added to this dictionary will be available in all templates:

```
<!-- templates/index.html -->
<h1>{{greeting}}</h1>
```

If the context needs to be different for each template, you can restrict contexts to certain templates by supplying `Site.make_site()` a sequence of regex-context pairs as the `contexts` keyword argument. When rendering a template, staticjinja will search this sequence for the first regex that matches the template's name, and use that context to interpolate variables. For example, the following code block supplies a context to the template named "index.html":

```
from staticjinja import Site

if __name__ == "__main__":
    context = {'knights': ['sir arthur', 'sir lancelot', 'sir galahad']}
    site = Site.make_site(contexts=[('index.html', context)])
    site.render()
```

```
<!-- templates/index.html -->
<h1>Knights of the Round Table</h1>
<ul>
{% for knight in knights %}
    <li>{{ knight }}</li>
{% endfor %}
</ul>
```

If contexts needs to be generated dynamically, you can associate filenames with functions that return a context ("context generators"). Context generators may either take no arguments or the current template as its sole argument. For example, the following code creates a context with the last modification time of the template file for any templates with an HTML extension:

```python
import datetime
import os

from staticjinja import Site


def date(template):
    template_mtime = os.path.getmtime(template.filename)
    date = datetime.datetime.fromtimestamp(template_mtime)
    return {'template_date': date.strftime('%d %B %Y')}

if __name__ == "__main__":
    site = Site.make_site(
        contexts=[('.*.html', date)],
    )
    site.render()
```

By default, staticjinja uses the context of the first matching regex if multiple regexes match the name of a template. You can change this so that staticjinja combines the contexts by passing `mergecontexts=True` as an argument to `Site.make_site()`. Note the order is still important if several matching regex define the same key, in which case the last regex wins. For example, given a build script that looks like the following code block, the context of the `index.html` template will be `{'title': 'MySite - Index', 'date': '05 January 2016'}`.

```python
import datetime
import os

from staticjinja import Site


def base(template):
    template_mtime = os.path.getmtime(template.filename)
    date = datetime.datetime.fromtimestamp(template_mtime)
    return {
        'template_date': date.strftime('%d %B %Y'),
        'title': 'MySite',
    }


def index(template):
    return {'title': 'MySite - Index'}

if __name__ == "__main__":
    site = Site.make_site(
        contexts=[('.*.html', base), ('index.html', index)],
        mergecontexts=True,
    )
    site.render()
```

### 1.2.4 Filters

Filters modify variables. staticjinja uses Jinja2 to process templates, so all the standard Jinja2 filters are supported. To add your own filters, simply pass `filters` as an argument to `Site.make_site()`.

```
filters = {
    'hello_world': lambda x: 'Hello world!',
    'my_lower': lambda x: x.lower(),
}

if __name__ == "__main__":
    site = Site.make_site(filters=filters)
    site.render()
```

Then you can use them in your templates as you would expect:

```
<!-- templates/index.html -->
{% extends "_base.html" %}
{% block body %}
<h1>{{'' | hello_world}}</h1>
<p>{{'THIS IS AN EXAMPLE WEB PAGE.' | my_lower}}</p>
{% endblock %}
```

### 1.2.5 Rendering rules

Rendering is the step in the build process where templates are evaluated to their final values using contexts, and the output files are written to disk.

Sometimes you'll find yourself needing to change how a template is rendering. For instance, you might want to render files with a `.md` from Markdown to HTML, without needing to put jinja syntax in your Markdown files. The following walkthrough is an explanation of the example that you can find and run yourself at `examples/markdown/`.

**Note:** If you want to run the example, you will need to install the `markdown` library from https://pypi.org/project/Markdown/

The structure of the project after running will be:

```
markdown
├── build.py
├── src
│   ├── _post.html
│   └── posts
│       ├── post1.md
│       └── post2.md
└── build
    └── posts
        ├── post1.html
        └── post2.html
```

First, look at `src/_post.html`...

```
<!-- src/_post.html -->
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>My Blog</title>
    </head>
    <body>
        {{ post_content_html }}
```

(continues on next page)

(continued from previous page)

```
    </body>
</html>
```

This template will be used for each of our markdown files, each of which might represent a blog post. This template expects a context with a `post_content_html` entry, which will get generated from each markdown file.

Now let's look at our build script. It does two things:

1. For every markdown template, use a context generator function (see above) to translate the markdown contents into html using the `markdown` library.
2. For each markdown template, compile that context into the `src/_post.html` template, and then write that to disk. The output `post1.html` should be placed in the same location relative to `out/` as the input `post1.md` was relative to `src/`.

The first step is accomplished in `md_context()`, and the second step is done in `render_md()`:

```
# build.py
#!/usr/bin/env python3
import os
import jinja2

# Markdown to HTML library
# https://pypi.org/project/Markdown/
import markdown

from staticjinja import Site

markdowner = markdown.Markdown(output_format="html5")
def md_context(template):
    with open(template.filename) as f:
        markdown_content = f.read()
        return {'post_content_html': markdowner.convert(markdown_content)}

def render_md(site, template, **kwargs):
    # Given a template such as posts/post1.md
    # Determine the post's title (post1) and it's directory (posts/)
    directory, fname = os.path.split(template.name)
    post_title, _ = fname.split(".")

    # Determine where the result will be streamed (build/posts/post1.html)
    out_dir = os.path.join(site.outpath, directory)
    post_fname = "{}.html".format(post_title)
    out = os.path.join(out_dir, post_fname)

    # Render and stream the result
    if not os.path.exists(out_dir):
        os.makedirs(out_dir)
    post_template = site.get_template("_post.html")
    post_template.stream(**kwargs).dump(out, encoding="utf-8")

site = Site.make_site(
        searchpath='src',
        outpath='build',
        contexts=[('.*.md', md_context)],
        rules = [('.*.md', render_md)],
        )
```

(continues on next page)

(continued from previous page)

```
site.render()
```

Note the rule we defined at the bottom. It tells staticjinja to check if the filename matches the `.*.md` regex, and if it does, to render the file using `render_md()`.

There are other, more complicated things you could do in a custom render function as well, such as not write the output to disk at all, but instead pass it somewhere else.

# CHAPTER 2

# API Documentation

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

## 2.1 Developer Interface

This part of the documentation covers staticjinja's API.

Most of staticjinja's functionality can be accessed with `Site.make_site()`, so pay particular attention to that.

### 2.1.1 Classes

**class** staticjinja.**Site**(*environment*, *searchpath*, *outpath*, *encoding*, *logger*, *contexts=None*, *rules=None*, *staticpaths=None*, *mergecontexts=False*)

The Site object.

**Parameters**

- **environment** – A `jinja2.Environment`.
- **searchpath** – A string representing the name of the directory to search for templates.
- **contexts** – A list of *regex, context* pairs. Each context is either a dictionary or a function that takes either no argument or or the current template as its sole argument and returns a dictionary. The regex, if matched against a filename, will cause the context to be used.
- **rules** – A list of *(regex, function)* pairs. The Site will delegate rendering to *function* if *regex* matches the name of a template during rendering. *function* must take a `staticjinja.Site` object, a `jinja2.Template`, and a context dictionary as parameters and render the template. Defaults to `[]`.
- **encoding** – The encoding of templates to use.
- **logger** – A logging.Logger object used to log events.

- **`staticpaths`** – Deprecated since version 0.3.4.

  List of directory names to get static files from (relative to searchpath).

- **`mergecontexts`** – A boolean value. If set to `True`, then all matching regex from the contexts list will be merged (in order) to get the final context. Otherwise, only the first matching regex is used. Defaults to `False`.

**`get_context`**(*template*)

Get the context for a template.

If no matching value is found, an empty context is returned. Otherwise, this returns either the matching value if the value is dictionary-like or the dictionary returned by calling it with *template* if the value is a function.

If several matching values are found, the resulting dictionaries will be merged before being returned if mergecontexts is True. Otherwise, only the first matching value is returned.

**Parameters** **`template`** – the template to get the context for

**`get_dependencies`**(*filename*)

Get a list of files that depends on the file named *filename*.

**Parameters** **`filename`** – the name of the file to find dependencies of

**`get_rule`**(*template_name*)

Find a matching compilation rule for a function.

Raises a `ValueError` if no matching rule can be found.

**Parameters** **`template_name`** – the name of the template

**`get_template`**(*template_name*)

Get a `jinja2.Template` from the environment.

**Parameters** **`template_name`** – A string representing the name of the template.

**`is_ignored`**(*template_name*)

Check if a template is an ignored template. Ignored templates are neither rendered nor used in rendering templates.

A template is considered ignored if it or any of its parent directories are prefixed with an `'.'`.

**Parameters** **`template_name`** – the name of the template to check

**`is_partial`**(*template_name*)

Check if a template is a partial template. Partial templates are not rendered, but they are used in rendering templates.

A template is considered a partial if it or any of its parent directories are prefixed with an `'_'`.

**Parameters** **`template_name`** – the name of the template to check

**`is_static`**(*template_name*)

Check if a template is a static template. Static template are copied, rather than compiled using Jinja2.

Deprecated since version 0.3.4.

A template is considered static if it lives in any of the directories specified in `staticpaths`.

**Parameters** **`template_name`** – the name of the template to check

**`is_template`**(*filename*)

Check if a file is a template.

A file is a considered a template if it is not partial, ignored, or static.

**Parameters** **`filename`** – the name of the file to check

**classmethod make_site**(*searchpath='templates'*, *outpath='.'*, *contexts=None*, *rules=None*, *encoding='utf8'*, *followlinks=True*, *extensions=None*, *staticpaths=None*, *filters={}*, *env_globals={}*, *env_kwargs=None*, *mergecontexts=False*)

Create a `Site` object.

**Parameters**

- **`searchpath`** – A string representing the absolute path to the directory that the Site should search to discover templates. Defaults to `'templates'`.

  If a relative path is provided, it will be coerced to an absolute path by prepending the directory name of the calling module. For example, if you invoke staticjinja using `python build.py` in directory `/foo`, then *searchpath* will be `/foo/templates`.

- **`outpath`** – A string representing the name of the directory that the Site should store rendered files in. Defaults to `'.'`.

- **`contexts`** – A list of *(regex, context)* pairs. The Site will render templates whose name match *regex* using *context*. *context* must be either a dictionary-like object or a function that takes either no arguments or a single `jinja2.Template` as an argument and returns a dictionary representing the context. Defaults to `[]`.

- **`rules`** – A list of *(regex, function)* pairs. The Site will delegate rendering to *function* if *regex* matches the name of a template during rendering. *function* must take a `staticjinja.Site` object, a `jinja2.Template`, and a context dictionary as parameters and render the template. Defaults to `[]`.

- **`encoding`** – A string representing the encoding that the Site should use when rendering templates. Defaults to `'utf8'`.

- **`followlinks`** – A boolean describing whether symlinks in searchpath should be followed or not. Defaults to `True`.

- **`extensions`** – A list of Jinja extensions that the `jinja2.Environment` should use. Defaults to `[]`.

- **`staticpaths`** – Deprecated since version 0.3.4.

  List of directories to get static files from (relative to searchpath). Defaults to `None`.

- **`filters`** – A dictionary of Jinja2 filters to add to the Environment. Defaults to `{}`.

- **`env_globals`** – A mapping from variable names that should be available all the time to their values. Defaults to `{}`.

- **`env_kwargs`** – A dictionary that will be passed as keyword arguments to the jinja2 Environment. Defaults to `{}`.

- **`mergecontexts`** – A boolean value. If set to `True`, then all matching regex from the contexts list will be merged (in order) to get the final context. Otherwise, only the first matching regex is used. Defaults to `False`.

**render**(*use_reloader=False*)

Generate the site.

**Parameters** **`use_reloader`** – if given, reload templates on modification

**render_template**(*template*, *context=None*, *filepath=None*)

Render a single `jinja2.Template` object.

If a Rule matching the template is found, the rendering task is delegated to the rule.

**Parameters**

- **template** – A `jinja2.Template` to render.
- **context** – Optional. A dictionary representing the context to render *template* with. If no context is provided, `get_context()` is used to provide a context.
- **filepath** – Optional. A file or file-like object to dump the complete template stream into. Defaults to to `os.path.join(self.outpath, template.name)`.

**render_templates**(*templates*, *filepath=None*)
Render a collection of `jinja2.Template` objects.

**Parameters**

- **templates** – A collection of Templates to render.
- **filepath** – Optional. A file or file-like object to dump the complete template stream into. Defaults to to `os.path.join(self.outpath, template.name)`.

**templates**
Generator for templates.

**class** staticjinja.**Reloader**(*site*)
Watches `site.searchpath` for changes and re-renders any changed Templates.

**Parameters** **site** – A *`Site`* object.

**event_handler**(*event_type*, *src_path*)
Re-render templates if they are modified.

**Parameters**

- **event_type** – a string, representing the type of event
- **src_path** – the path to the file that triggered the event.

**should_handle**(*event_type*, *filename*)
Check if an event should be handled.

An event should be handled if a file in the searchpath was modified.

**Parameters**

- **event_type** – a string, representing the type of event
- **filename** – the path to the file that triggered the event.

**watch**()
Watch and reload modified templates.

# CHAPTER 3

# Contributor Guide

If you want to contribute to the project, this part of the documentation is for you.

## 3.1 Contributing

staticjinja is under active development, and contributions are more than welcome!

### 3.1.1 Get the Code

staticjinja is actively developed on GitHub, where the code is always available.

You can clone the public repository:

```
git clone git://github.com/Ceasar/staticjinja.git
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

### 3.1.2 How to Help

1. Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug.
2. Fork the repository on GitHub to start making your changes to the **master** branch (or branch off of it).
3. Send a pull request and bug the maintainer until it gets merged and published. :) Make sure to add yourself to AUTHORS.

## 3.2 Authors

staticjinja was written and maintained by Ceasar Bautista and various contributors. Now Caesar has passed maintaining responsibilities to Nick Crews. If you would like to share the responsibilities of maintaining, let Nick know, he often does not get to issues as soon as he would like :)

### 3.2.1 Development Lead

- (Current lead) Nick Crews, @NickCrews, <nicholas.b.crews@gmail.com>
- (Former lead) Ceasar Bautista, @Caesar, <cbautista2010@gmail.com>

### 3.2.2 Patches and Suggestions

- Dominic Rodger (dominicrodger)
- Filippo Valsorda
- Alexey (rudyrk)
- Jacob Lyles
- Jakub Zalewski
- NeuronQ
- Eduardo Rivas (jerivas)
- Bryan Bennett (bbenne10)
- Anuraag Agrawal (anuraaga)
- saschalalala
- Tim Best (timbest)
- Fasih Ahmad Fakhri

## 3.3 Changelog

### 3.3.1 0.4.0

- Improve Travis CI testing: Add Windows and OSX, stop testing python2, add newer python3 versions, update tox.ini.
- Convert all print()s to logger.logs().
- Make CLI interface use Site.make_site() instead of deprecated make_site().
- Simplify style and how kwargs are passed around.
- Single-source the version info so it's always consistent.
- Minor fixes, updates, improvements to README, AUTHORS, CONTRIBUTING, setup.py, __init__.py docstring,
- Rename Site._env to Site.env, making it publicly accessible, for instance in custom rendering functions.

- Fix docstring for the expected signature of custom rendering rules so they expect a staticjinja.Site as opposed to a jinja2.Environment
- Make is_{template,static,ignored,partial} functions be consistent with taking template names(always use `/`), not file names (use os.path.sep), making them consistent between OSs. https://github.com/staticjinja/staticjinja/issues/88
- Update and improve docs, add .readthedocs.yml so that ReadTheDocs.org can automatically pull from the repo and build docs on changes. Add a badge for if the doc build passes. Add readthedocs build task as a GitHub check, so new PRs and branches will automatically get this check.
- Change single example/ directory to a collection of examples in examples/, and add in an example for using custom rendering rules to generate HTML from markdown. This also fixes the totally wrong tutorial on the docs for how to use custom rendering rules. See https://github.com/staticjinja/staticjinja/pull/102
- Update dependencies using pip-tools to automatically generate indirect dependencies from direct dependencies:
  - jinja2==2.6 -> jinja2==2.11.2
  - argh==0.21.0 -> REMOVED
  - argparse==1.2.1 -> REMOVED
  - docopt==0.6.1 -> docopt==0.6.2
  - easywatch==0.0.5 -> easywatch==0.0.5
  - pathtools==0.1.2 -> pathtools==0.1.2
  - watchdog==0.6.0 -> watchdog==0.10.3
  - wsgiref==0.1.2 -> REMOVED
  - NONE -> markupsafe==1.1.1

### 3.3.2 0.3.5

- Make README less verbose.
- Only warn about using deprecated `staticpaths` if `staticpaths` is actually used.
- Updated easywatch to 0.0.5

### 3.3.3 0.3.4

- Move `make_site()` to `Site.make_site()`.
- Deprecate `staticpaths` argument to `Site()` and `Site.make_site()`. See Issue #58.
- Add an option (default `True`) for Jinja's `FileSystemLoader` follow to symlinks when loading templates.
- Ensure that the output directory exists, regardless of whether custom rendering rules were supplied. Before that was only ensured if custom rendering rules were not given.
- License file is included now in distributions.
- Add documentation for partial and ignored files.
- Updated easywatch to 0.0.4.
- Fix a few style errors.

### 3.3.4 0.3.3

- Enable users to direct pass dictionaries instead of context generator in Site and make_site() for contexts that don't require any logic.
- Introduces a `mergecontexts` parameter to Site and make_site() to direct staticjinja to either use all matching context generator or only the first one when rendering templates.

### 3.3.5 0.3.2

- Allow passing keyword arguments to jinja2 Environment.
- Use `shutil.copy2` instead of `shutil.copyfile` when copying static resources to preserve the modified time of files which haven't been modified.
- Make the Reloader handle "created" events to support editors like Pycharm which save by first deleting then creating, rather than modifying.
- Update easywatch dependency to 0.0.3 to fix an issue that occurs when installing easywatch 0.0.2.
- Make `--srcpath` accept both absolute paths and relative paths.
- Allow directories to be marked partial or ignored, so that all files inside them can be considered partial or ignored. Without this, developers would need to rename the contents of these directories manually.
- Allow users to mark a single file as static, instead of just directories.

### 3.3.6 0.3.1

- Add support for filters so that users can define their own Jinja2 filters and use them in templates:

```
filters = {
    'filter1': lambda x: "hello world!",
    'filter2': lambda x: x.lower()
}
site = staticjinja.make_site(filters=filters)
```

- Add support for multiple static directories. They can be passed as a string of comma-separated names to the CLI or as a list to the Renderer.
- "Renderer" was renamed to "Site" and the Reloader was moved staticjinja.reloader.

### 3.3.7 0.3.0

- Add a command, `staticjinja`, to handle the simple case of building context-less templates.
- Add support for copying static files from the template directory to the output directory.
- Add support for testing, linting and checking the documentation using `tox`.

### 3.3.8 0.2.0

- Add a `Reloader` class.
- Add `Renderer.templates`, which refers to the lists of templates available to the `Renderer`.
- Make `Renderer.get_context_generator()` private.

- Add `Renderer.get_dependencies(filename)`, which gets every file that depends on the given file.
- Make `Renderer.render_templates()` require a list of templates to render, *templates*.

# Python Module Index

# Index