Complexity Documentation

Release 0.9.1

Audrey Roy

October 14, 2016

Contents

CHAPTER 1

A refreshingly simple static site generator, for those who like to work in HTML.

This friendly guide contains everything you need to know to create and publish static HTML websites with Complexity.

1.1 Complexity

A refreshingly simple static site generator, for those who like to work in HTML.

1.1.1 Documentation

The full documentation is at http://complexity.rtfd.org.

1.1.2 Quickstart

Try it out:

```
$ pip install complexity
$ git clone git@github.com:audreyr/complexity-example.git my_proj
$ cd my_proj
$ complexity project/ www/
```

Once you've done that, open a web browser to http://127.0.0.1:9090 to see the newly generated Complexity static site.

1.1.3 Features

- Works on Python 2.6, 2.7, and 3.3, and on PyPy.
- Takes simple HTML templates as input.
- Data from .json files turns into template context data.
- Template inheritance, filters, etc. (Brought to you by Jinja2.)
- Auto-expands .html file URLs into cleaner URLs (e.g. about.html gets expanded to /about/)
- Can optionally be used as a library instead of from the command line. See Using Complexity as a Library for details.

1.1.4 Best Used With

Complexity is designed to be used with these packages:

- Simplicity: Converts ReStructuredText into JSON, which Complexity can use as input.
- A Lot of Effort: Deploys a static website (e.g. the output of Complexity) to Amazon S3.
- Cookiecutter: Creates projects from project templates.

Sure, they could have all been built into Complexity, but decoupling them seemed like a nice thing to do.

1.1.5 Community

- Stuck? Don't know where to begin? File an issue and we'll help you.
- We love contributions. Read about how to contribute.

1.2 Installation

Note: Mac users may need to use "sudo" before the install commands. But use virtualenv if you don't want to sudo – it's great.

1.2.1 Best Method: Pip

This will download and install Complexity:

```
$ pip install complexity
```

This method requires an installer tool called *pip*, which you can get from http://www.pip-installer.org/.

Don't worry, you can later uninstall Complexity like this:

```
$ pip uninstall complexity
```

1.2.2 Alternate Method 1: Setup.py

If you can't use *pip* to install Complexity, download the latest Complexity release from https://pypi.python.org/pypi/complexity.

Then unzip and install Complexity:

```
$ tar xzvf <name of file>
$ cd <name of unzipped dir>
$ python setup.py install
```

1.2.3 Alternate Method 2: Easy Install

If neither of the above methods work for some reason, try this:

```
$ easy_install complexity
```

And if that doesn't work, see Troubleshooting.

1.3 Upgrading

Note: Mac users may need to use "sudo", but try it without the "sudo" first. (Or was that "sussudio"? "Su-su-sussudiooo!!" But use virtualenv if you don't want to sudo.)

1.3.1 How to Upgrade From an Earlier Version

To upgrade Complexity:

```
$ pip install -U complexity
```

Or if using easy_install:

```
$ easy_install --upgrade complexity
```

1.3.2 Things to Know Before Upgrading

Some releases may require you to make changes on your end; if so, instructions will be described in History.

Of course, if you run into any problems and need help, file an issue with details so someone can help.

1.4 Tutorial

1.4.1 Part 0: Overview

This is the directory structure for a minimal Complexity site:

```
my_repo/
-- project/       <--------- input
|   -- assets/
|   |   -- css/
|   |   -- js/
|   |   -- img/
|   |
|   -- templates/
|       -- base.html
|       -- index.html
|       -- about.html
|
-- www/          <---------- output
    -- index.html
    -- about/
    |   -- index.html
    -- css/
    -- js/
    -- img/
```

1.4.2 Part 1: Setup

First, grab a copy of the example Complexity site:

```
git clone https://github.com/audreyr/complexity-example.git
```

Open everything in a text editor. You should see a main *project/* directory with subfolders for your work:

- Study the template files in *templates/*. We'll go over them shortly.
- Notice the *assets/* directory. That is where you put your static files.
- Creating additional directories in *assets/* (e.g. *ico/*) is fine; they'll get copied over to *www/* without modification.

At the same level as *project/*, a *www/* directory will be auto-generated. It will contain your final rendered templates and optimized static assets.

When you're done, you should have a project structure like that in https://github.com/audreyr/complexity-example.

1.4.3 Part 2: What's in a Complexity Site?

Here's what a very simple Complexity site looks like:

project/templates/base.html:

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}{% endblock %} - Built with Complexity</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap -->
    <link href="/css/bootstrap.min.css" rel="stylesheet" media="screen">
</head>
<body>
    <div class="container">
        <div class="navbar">
            <div class="navbar-inner">
                <a class="brand" href="#">Complexity</a>
                <ul class="nav">
                    <li><a href="/">Home</a></li>
                    <li><a href="/about/">About</a></li>
                </ul>
            </div>
        </div>

        {% block content %}
        {% endblock %}
    </div>

<script src="http://code.jquery.com/jquery.js"></script>
<script src="/js/bootstrap.min.js"></script>
</body>
</html>
```

project/templates/index.html:

```
{% extends "base.html" %}

{% block title %}Home{% endblock %}
```

```
{% block content %}
<div class="row">
    <div class="span12">
        <h1>Home</h1>
        <p>This is the Home page of your website.</p>
    </div>
</div>
{% endblock %}
```

project/templates/about.html:

```
{% extends "base.html" %}

{% block title %}About{% endblock %}

{% block content %}
<div class="row">
    <div class="span12">
        <h1>About</h1>
        <p>This is the About page of your website.</p>
    </div>
</div>
{% endblock %}
```

Notice how *index.html* and *about.html* both share a common parent template, *base.html*.

1.4.4 Part 3: Generate the Site and Serve It Locally

Run the *complexity* command, passing it input and output directories:

```
$ complexity project/
```

This results in the following:

- A *www/* directory gets created, containing your generated static HTML site.
- Templates are rendered and output to files smartly:
  - Any templates starting with "base" are assumed to be parent templates and not rendered on their own (e.g. *base.html*, *base_section.html*)
  - Templates named *index.html* are output to the same corresponding locations in *www/*.
  - Other templates are expanded in order to hide the ".html" extension. For example, *about.html* is expanded to *about/index.html*.
- A lightweight server starts up locally, serving your site so that you can see how it looks and check that everything works.

Open a web browser to http://127.0.0.1:9090. You should see your newly generated site!

In an upcoming release, the following will also occur during Complexity's generation process:

- CSS will be minified and concatenated.
- SCSS and/or LESS will compiled to CSS, then minified and concatenated.
- JS will minified, concatenated, and obfuscated.

Development is happening at a rapid pace, so stay tuned. To keep updated, watch and star https://github.com/audreyr/complexity on GitHub.

1.4.5 Part 4: Upload the Site to Amazon S3

For site deployment we'll use the "alotofeffort" tool. It is designed for use with Complexity, but it works with non-Complexity sites just as well.

Install it:

```
$ pip install alotofeffort
```

Save the following in *~/.boto*:

```
[Credentials]
aws_access_key_id = ...
aws_secret_access_key = ...
```

Replace ... with your AWS access credentials, of course.

Then deploy the *www/* directory to any S3 bucket that you own:

```
$ alotofeffort www/ your-s3-bucketname
```

Your site is now live! Go to the URL that *alotofeffort* prints out after it finishes uploading.

Point your domain name at that URL, and you'll be done.

1.5 Advanced Usage

In the tutorial, you saw an example of a minimal Complexity project layout. Now here is an example of a more advanced Complexity site:

```
my_repo/
-- project/       <--------- input
|   -- assets/
|   |   -- css/
|   |   -- js/
|   |   -- img/
|   |   -- ico/
|   |   -- robots.txt
|   -- context/
|   |   -- books.json
|   |   -- movies.json
|   -- templates/
|   |   -- base.html
|   |   -- index.html
|   |   -- about.html
|   -- complexity.yml
|
-- www/          <---------- output
    -- index.html
    -- about/
    |   -- index.html
    -- css/
    -- js/
    -- img/
    -- ico/
```

Let's explore some of Complexity's advanced features.

1.5.1 Config Using complexity.yml

You can configure a Complexity project with a *complexity.yml* file like this:

```
# Config file for a Complexity project

# Directories are relative to current (project) dir
templates_dir: "templates"
assets_dir: "assets"
context_dir: "context"
output_dir: "../www"

# List of templates that should not be expanded to pretty-format URLs
unexpanded_templates:
 - "404.html"
 - "500.html"
```

Put *complexity.yml* in your project root (e.g. in project/).

Here is what you can configure:

- *templates_dir*: Directory containing templates. Anything that needs to be templated goes here.
- *assets_dir*: Directory containing static assets (to be copied over without templating).
- *context_dir*: Directory containing *.json* files to be turned into context variables for the templates.
- *output_dir*: Directory where the generated website will be output.
- *unexpanded_templates*: List of HTML templates for which you want to keep the URLs unexpanded (e.g. *404.html* instead of *404/index.html*).

All of the above are optional.

Complexity uses sensible defaults. If you don't specify a *complexity.yml*, this is the assumed default config:

```
templates_dir: "templates"
assets_dir: "assets"
context_dir: "context"
output_dir: "../www"
```

1.5.2 JSON Auto-Loading

Data from .json files in your context directory automatically turns into template context data.

For example, suppose you have this in *context/books.json*:

```
[
  {
    "url": "http://www.amazon.com/Two-Scoops-Django-Best-Practices/dp/1481879707/",
    "title": "Two Scoops of Django"
  },
  {
    "url": "http://www.amazon.com/Very-Magical-Caterpillar-Tale-Butterfly/dp/1453714081/",
    "title": "A Very Magical Caterpillar Tale"
  }
]
```

Then you can refer to the books in a template like this:

```
{% extends 'base.html' %}

{% block title %}Index{% endblock %}

{% block content %}
    <p>Here are my books:</p>
    {% for book in books %}
        <a href="{{ book.url }}">{{ book.title }}</a>
    {% endfor %}
{% endblock %}
```

The contents of *books.json* get turned into *{{ books }}*, which in this case is a list that you can iterate over.

What About Static JSON Files?

If you have .json files that you want served as static assets rather than turned into context data, that's fine.

Just put them in *assets/js/* (or anywhere in *assets/*), and they'll get copied over to the output directory like any other static asset.

1.5.3 Other Asset Directories and Files

You can create any type of asset directory or file that you want in *assets/* (or your desired assets directory).

All assets will get copied over to *www/* when you generate your site.

Note: Better handling/processing of assets will be implemented in an upcoming release, including CSS/JS minification, image optimization, and SASS and/or LESS compilation.

1.5.4 Using Complexity as a Library

Complexity can be used just like any other Python package.

You can simply call the Complexity API like this:

```
from complexity.main import complexity

complexity('project/', 'www/')
```

Calling other Complexity API functions is just as straightforward:

```
from complexity import generate

# Optionally generate context if you need to
context = generate_context(context_dir='project/context/')

# Generate HTML from your templates (and context, if you have it)
generate.generate_html(templates_dir='project/templates/', output_dir='www/', context=context)

# Copy assets over
generate.copy_assets(assets_dir='project/assets/', output_dir='www/')
```

This allows you to use Complexity as a dependency in your own Python projects.

Note: As of this release, the API works, but it is subject to change. Please pin your dependencies if you need this to be stable, and keep an eye on this section for changes when you upgrade.

1.6 Troubleshooting

1.6.1 Installation Problems

Problem: Pip Fails

Don't worry if *pip* fails like this:

```
$ pip install complexity
...
error: could not create '/Library/Python/2.7/site-packages/complexity':
Permission denied
```

We've got a couple of solutions for that.

Best Solution: Use Virtualenv

1. Install virtualenv systemwide with *pip*:

```
$ sudo pip install virtualenv
```

2. Create a virtualenv for Complexity:

```
$ virtualenv complexity-env
$ source complexity-env/bin/activate
        (or complexity-env/Scripts/activate.bat on Windows)
(complexity-env) $
```

3. Install Complexity into the virtualenv:

```
(complexity-env) $ pip install complexity
```

Alternate Solution: Install Systemwide

1. Install Complexity systemwide with *pip*:

```
$ sudo pip install complexity
```

2. If that doesn't work, you can use *easy_install* instead:

```
$ sudo easy_install complexity
```

1.6.2 Site Generation Problems

Problem: Site Generation Fails

If you get an error like this:

```
jinja2.exceptions.TemplateSyntaxError: Unexpected end of template. Jinja
was looking for the following tags: 'endblock'. The innermost block that
needs to be closed is 'block'.
```

Then check your templates carefully and make sure that you've closed all blocks properly with *{% endblock %}*.

1.6.3 Still Having Problems?

File an issue here with the following info:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the problems.

CHAPTER 2

API Reference

2.1 complexity Package

2.1.1 `complexity` Package

complexity

Main package for Complexity.

2.1.2 `conf` Module

complexity.conf

Functions for reading a *complexity.yml* configuration file and doing various configuration-related things.

`complexity.conf.`**`get_unexpanded_list`**(*conf_dict*)
Given a configuration dict, returns the list of templates that were specified as unexpanded.

`complexity.conf.`**`read_conf`**(*directory*)
Reads and parses the *complexity.yml* configuration file from a directory, if one is present. :param directory: Directory to look for a *complexity.yml* file. :returns: A conf dict, or False if no *complexity.yml* is present.

2.1.3 `exceptions` Module

complexity.exceptions

All exceptions used in the Complexity code base are defined here.

exception `complexity.exceptions.`**`ComplexityException`**
Bases: `Exception`

Base exception class. All Complexity-specific exceptions subclass *ComplexityException*.

exception `complexity.exceptions.`**`MissingTemplateDirException`**
Bases: *`complexity.exceptions.ComplexityException`*

Raised when a project is missing a *templates/* subdirectory.

exception complexity.exceptions.**NonHTMLFileException**
Bases: *complexity.exceptions.ComplexityException*

Raised when a project's *templates/* directory contains a non-HTML file.

exception complexity.exceptions.**OutputDirExistsException**
Bases: *complexity.exceptions.ComplexityException*

Raised when a project's output_dir exists and no_input=True.

2.1.4 generate Module

2.1.5 main Module

2.1.6 prep Module

complexity.prep

Functions for preparing a Complexity project for static site generation, before it actually happens.

complexity.prep.**prompt_and_delete_cruft**(*output_dir*)
Asks if it's okay to delete *output_dir/*. If so, go ahead and delete it.

Parameters **output_dir** (*directory*) – The Complexity output directory, e.g. *www/*.

2.1.7 serve Module

complexity.serve

Functions for serving a static HTML website locally.

complexity.serve.**serve_static_site**(*output_dir*, *port=9090*)
Serve a directory containing static HTML files, on a specified port.

Parameters **output_dir** – Output directory to be served.

2.1.8 utils Module

complexity.utils

Helper functions used throughout Complexity.

complexity.utils.**make_sure_path_exists**(*path*)
Ensures that a directory exists.

Parameters **path** – A directory path.

complexity.utils.**query_yes_no**(*question*, *default='yes'*)
Ask a yes/no question via *raw_input()* and return their answer.

Parameters

- **question** – A string that is presented to the user.
- **default** – The presumed answer if the user just hits <Enter>. It must be "yes" (the default), "no" or None (meaning an answer is required of the user).

The "answer" return value is one of "yes" or "no".

Adapted from http://stackoverflow.com/questions/3041986/python-command-line-yes-no-input http://code.activestate.com/recipes/577058/

complexity.utils.**unicode_open**(*filename*, **args*, ***kwargs*)

Opens a file as usual on Python 3, and with UTF-8 encoding on Python 2.

Parameters **filename** – Name of file to open.

CHAPTER 3

Project Info

3.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/audreyr/complexity/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" is open to whoever wants to implement it.

Write Documentation

Complexity could always use more documentation, whether as part of the official Complexity docs, in docstrings, or even on the web in blog posts, articles, and such.

Create Examples

Some examples of real Complexity sites, whether open-source or closed-source, would be awesome.

If you create an example Complexity site, file an issue so that it can be linked from the docs.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/audreyr/complexity/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.1.2 Get Started!

Ready to contribute? Here's how to set up *complexity* for local development.

1. Fork the *complexity* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/complexity.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv complexity
$ cd complexity/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 complexity tests
    $ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and PyPy. Check https://travis-ci.org/audreyr/complexity/pull_requests and make sure that the tests pass for all supported Python versions.

3.1.4 Tips

To run a particular test:

```
$ python -m unittest tests.test_complexity.TestComplexity.test_make_sure_path_exists
```

To run a subset of tests:

```
$ python -m unittest tests.test_complexity
```

3.2 Credits

3.2.1 Development Lead

- Audrey Roy (@audreyr)

3.2.2 Contributors

- Daniel Greenfeld (@pydanny)
- Marko Mrdjenovic (@friedcell)
- Alja Isakovic (@ialja)

3.2.3 Special Thanks

- Daniel Greenfeld greatly helped during Complexity's initial development and came up with the name.
- Complexity Sphinx theme is a heavily customized version of Kenneth Reitz's Requests theme, which itself is a modified version of Armin Ronacher's Flasky theme.

3.3 History

3.3.1 0.9.1 (2013-12-02)

- Depend on Jinja2 >= 2.4, not == 2.7.

3.3.2 0.9.0 (2013-08-28)

- CONFIG CHANGE: Configuration is now via a *complexity.yml* file inside the project, instead of a *complexity.json* file.
- Support for an *unexpanded_templates* config option (#23).
- Support for non-HTML files in *templates/* (or whatever you set *templates_dir* to be).

See http://complexity.readthedocs.org/en/latest/advanced_usage.html#config-using-complexity-yml for more info.

3.3.3 0.8.0 (2013-08-10)

- USAGE CHANGE: At the command line, Complexity no longer takes an output_dir argument. It now assumes that your output_dir is *www/* by default, but you can customize it in *complexity.json*.
- Support for configuration via *complexity.json*: you can specify any or all of the following key/value pairs:
  - *output_dir*
  - *templates_dir*
  - *assets_dir*
  - *context_dir*

See http://complexity.readthedocs.org/en/latest/advanced_usage.html#config-using-complexity-json for more info.

3.3.4 0.7 (2013-08-05)

A couple of small but important renames. If you rely on either of the following defaults, you will need to rename them in your Complexity project.

- Directory parameter for .json files to be turned into context data has been renamed from *json_dir* to *context_dir*.
- Default context directory value *json/* has been changed to *context/*.

Sometimes you want your .json files to be turned into context variables, and sometimes you don't. This rename alleviates confusion when working with non-context .json files.

3.3.5 0.6 (2013-07-26)

- Support for multi-level template directories. (Upgrade to at least 0.6 if you want to have folders within folders and beyond in *templates/*.)
- Skip non-HTML files in *templates/* rather than raising *NonHTMLFileException*.

3.3.6 0.5 (2013-07-25)

- Improved static site generation API - better parameters are used.
- Files in the root of *assets/* (or the asset directory) now get copied over to the output.
- Much more documentation.

3.3.7 0.4.2 (2013-07-21)

- Make reading of JSON files from *json/* optional.

3.3.8 0.4.1 (2013-07-19)

- Fix reading of JSON files from *json/*.

3.3.9 0.4 (2013-07-19)

- Project layout is now:

```
my_repo/
-- project/       <--------- input
|   -- assets/
|   |   -- css/
|   |   -- js/
|   |   -- img/
|   -- json/
|   |   -- stuff.json
|   -- templates/
|       -- base.html
|       -- index.html
|       -- about.html
-- www/          <---------- output (generated)
    -- index.html
    -- about/
    |   -- index.html
    -- css/
    -- js/
    -- img/
```

- Assets are copied over to *www/* during site generation.
- If the *www/* directory was previously created, it prompts the user and then deletes it before site regeneration.
- Templates starting with *base* are not generated as individual pages. They are meant to be extended in other templates.

3.3.10 0.3 (2013-07-18)

- Graceful shutdown/restart of dev server.
- Required input and output dir arguments.
- Optional port argument.
- Improved server start/stop messages.
- Major internal refactor.

3.3.11 0.2.1 (2013-07-15)

- Fixes to setup.py.

3.3.12 0.2.0 (2013-07-15)

- Data from .json files now gets read as template context data.
- Tested (and passing!) on Python 2.6, 2.7, 3.3, PyPy.

3.3.13 0.1.1 (2013-07-10)

- First release on PyPI.

CHAPTER 4

Index

- genindex
- modindex

C

C

G

M

N

O

P

Q

R

S

U