Taika Documentation

Release 0.6.0

Hector Martinez-Lopez

Mar 06, 2020

Contents

CHAPTER 1

Taika Documentation

Taika is a simple Static Content Generator, it can be a simple files processor or can be more complex, if extensions are used. It aims to be simple and extensible, so if a user needs more functionality can add it via extensions.

You want a full overview of the documentation? Take a look at the *Table of Contents*.

1.1 Getting started

- *Installation*
- *Basic Tutorial*

1.2 How-To Guides

Take a look at our *Guides* to build more complex things with Taika.

1.3 Extensions

Add more functionality to your site with extensions!

Check the builtin extensions. Here are a few of them:

- *taika.ext.layouts – Jinja layouts*
- *taika.ext.rst – ReStructuredText*
- *taika.ext.collections – Grouping content*

Or they don't fit your needs, *create your own extension*.

1.4 API reference

Check the *configuration reference*.

How Taika works? Take a look at the *Reference*.

1.5 Taika itself

How is Taika managed and how you can contribute to it.

Contributing to Taika

CHAPTER 2

Introduction

Take a look here if you are a newcomer!

2.1 Installation

2.1.1 From PyPI

To install Taika, run in your terminal:

```
pip install taika
```

You will get the last release available from the PyPI.

2.1.2 From source

The sources for Taika can be downloaded from the GitLab repo.

Cloning using `git`:

```
git clone git://gitlab.com/hectormartinez/taika.git
```

Or downloading a tar:

```
curl -OL https://gitlab.com/hectormartinez/taika/repository/master/archive.tar
pip install archive.tar
```

Once you have a copy of the source, you can install it with:

```
cd taika
pip install .
```

2.2 Basic Tutorial

Create a directory with files:

```
mkdir source
touch source/index.txt
touch source/contents.rst
touch source/first-post.md
touch source/taika.yml
```

And run the `taika` command:

```
taika source /tmp/taika/
ls /tmp/taika/
```

Taika will get your files from `source` and will process and write them into `/tmp/taika`:

```
contents.rst  first-post.md  index.txt  taika.yml
```

The extensions that you load into Taika will modify those files and probably will have different extensions, new content, etc. Take a look at the *extensions*.

CHAPTER 3

How-To Guides

Here you will find recipes to do things with Taika!

3.1 Blogging site

With taika you can setup a little static blog as you can do it with Jekyll, Hugo, Pelican or others. You should add a few extensions and options to your `taika.yml` configuration.

```
title: Hector Martinez-Lopez
subtitle: Scientific Software Developer
url: https://hectormartinez.gitlab.io
extensions:
   - taika.ext.rst
   - taika.ext.collections
   - taika.ext.layouts
   - taika.ext.excerpt

assets:
   source: assets
   destination: assets

collections:
   posts:
      pattern: "posts/(?!index.rst).*"

layouts_pattern: '*.rst'
layouts_options:
   autoescape: False
   lstrip_blocks: True
   trim_blocks: True
```

Adding those options, Taika will be able to parse the '*.rst' files into html, aggregate them into collections, render them using Jinja2 and extract their excerpt. You need to have a `templates/` directory in the working directory so Jinja2 can template your posts and pages. Check the *layouts extension* for more detail.

3.2 Create your own extension

If you want to create yourself an extension, this is the place. You can take a look at the source of builtin extensions such as *layouts* or *rst*.

3.2.1 Available events to register to

The following are the different events which you can register your extension to.

Note: The arguments that are listed are given as *arguments*, not *keyword_arguments*.

doc-post-read

Arguments: `site, document`

Called after each document read.

site-post-read

Arguments: `site`

Called after all the documents have been read.

3.2.2 Empty extension

This will be the backbone of your extension, one that does nothing:

```
def do_nothing(site, document):
   pass

def setup(site):
   pass
```

Taika will use the `setup()` as entry point, so you should do your loading there.

Save it as a Python file and place it in the *extensions* and put it's path into `extensions_path`, so we can find it and load it:

```
# taika.yml
...
extensions_path: /path/to/my_extensions
extensions:
   ...
   - my_extension  # <--
   ...
...
```

3.2.3 Adding functionality

Here, we use as example an extension that will print certain documents. First we print all the documents:

```
def print_document(site, document):
    print(document)

```

(continues on next page)

(continued from previous page)

```
def setup(site):
    site.events.register("doc-post-read", print_document)
```

Note that we are registering our function using `site.events.register`.

3.2.4 Making the extension configurable

Now we want to print only certain documents, based on the frontmatter keys. But we want to configure which key is needed to print the documents, so we make our extension to read `site.config`.:

```
def print_document(site, document):
    on_key = site.config.get("print_document_on_key", DEFAULT_KEY)
    if on_key in document:
        print(document)

def setup(site):
    site.events.register("doc-post-read", print_document)
```

We used the `get` method on the configuration and we passed our key name and a default value. Later, we decide that we want to pass a list of keys which will trigger the print. We modify our extensions as follows:

```
def print_document(site, document):
    on_keys = site.config.get("print_document_on_keys", DEFAULT_KEYS)
    match = [key in document for key in on_keys]
    if any(match):
        print(document)

def setup(site):
    site.events.register("doc-post-read", print_document)
```

And that's all! Be creative!

CHAPTER 4

Extensions

Taika comes with builtin extensions that will help you to develop better your site.

Take a look at the builtin extensions, or create your own!

4.1 `taika.ext.collections` – Grouping content

This extension groups documents using patterns specified by the user. It also order those documents using certain keys specified by the user.

It uses the patterns listed from top to bottom, the documents not included in the first pattern are not matched against the second pattern, so be liberal in the first pattern and more restricted at the bottom. Also, if the pattern starts with `!` (exclamation mark) the documents matching will be excluded. For example:

```
collections:
posts:
  patterns:
    - "posts/*" # Include all under posts
    - "!posts/index.rst" # Ignore posts/index.rst
```

4.1.1 Event

This extension is subscribed to the `site-post-read` event.

4.1.2 Process

1. Setup where the `collections` keys is retrieved.
2. When the extension is called, scans the documents checking their `path`.
3. If `path` matches the patterns provided, it's added to the collection.

4. Finally, the attribute `collections` is created on `taika.Taika`.

4.1.3 Configuration

```
# Match all but the index.rst file on posts/
collections:
  posts:
    patterns:
      - "posts/*" # Include all under posts
      - "!posts/index.rst" # Ignore posts/index.rst
```

collections (*dict*)
Default: {}

A dictionary where each key specifies the name of the collection.

collection.**patterns** (*list*)
Default: [‘’] (empty string)

The patterns to be used in order to group the files. By default, it matches nothing.

4.1.4 Classes and Functions

class Collector (*config*)
Main class which retrieves the configuration and the organize the documents.

organize (*self*, *site*)
Classify the documents and creates the collections attribute on *site*.

match (*path*, *patterns*, *reverse_character=’!’*)

setup (*site*)

4.2 `taika.ext.excerpt` – Documents excerpts

This extensions creates a excerpt for the documents based on it’s content.

4.2.1 Event

This extension is subscribed to the “doc-post-read” event.

4.2.2 Frontmatter

excerpt_separator (*str*)
Use this separator instead of the global separator defined in the configuration.

4.2.3 Configuration

```
excerpt_separator: <!-- read-more -->
```

excerpt_separator (*str*)
Default: `None`

A string that will be used as excerpt separator. Default to `None` so no excerpt will be generated.

4.2.4 Process

1. Check for the frontmatter option, otherwise use the global or the default separator.
2. If separator is None, the first `<p>` tag is retrieved if existent.
3. If the first `<p>` tag is not found, `\n\n` (double line separator) is used as separator.
4. If separator is something, the text before that separator is retrieved if existent.
5. The `excerpt` is inserted into the document so it will be accessible.

4.2.5 Classes and Functions

get_excerpt (*site*, *document*)

setup (*site*)

4.3 `taika.ext.layouts` – Jinja layouts

This extension renders documents content trought a the Jinja2 templating engine. It also renders the content of documents itself if any Jinja block/comment/var is detected.

4.3.1 Event

This extension is subscribed to the `site-post-read` event.

4.3.2 Payload

When the content and the templates are rendered, certain payload is passed and becomes accessible by both content and templates. This payload has two main keys: `site` and `document`.

Using `document` you can access the document attributes being processed, such as the `path`, `content`, etc. Check *Document specification* for details.

Inside `site`, the *`taika.Taika`* is accessible.

Note: Note that `site.config` returns a dictionary with all the sections included, so to access which extensions are listed you should use `site.config.taika.extensions`. This is a long "import-like" statement, and probably we will shrink it in the future.

4.3.3 Frontmatter

layout (*str*)
The layout that should render the document. Should exist under the `layouts_path`. If None the documents is not passed throught the template, but its body is still rendered.

4.3.4 Configuration

Note: All configuration hangs from a key in the YML configuration named 'layouts'. Inside it, you can add the following options:

`path` (*list*)
Default: **[./templates/]**

A list of paths from where the layouts will be loaded.

`options` (*dict*)
Default: **{}** (empty-dict)

A dictionary (key-value) of options to pass to the Jinja environment when created.

`default` (*str*)
Default: **index.html**

The default layout if the document has no `layout` defined.

`patterns` (*str*)
Default: **["*"]**

A list of patterns to match Which files should be renderered. Default to all the files.

4.3.5 Default filters

`link`
Link against other documents inside your site using `{{ '/posts/2019/my-other-post.md' | link }}`. Relative links not supported, only absolute paths will be accepted.

4.3.6 Process

1. (pre-registering) The *`JinjaRenderer`* is initialized with the configuration. The Jinja environment is created and the templates loaded.
2. Checks if the path of the document matches `layouts_pattern`, if not, skips it.
3. Composes the layout using the document itself, so the document metadata is available directly.
4. If the content has any Jinja flag, it is renderered, so you can include Jinja syntax into the document text.
5. Then the content (rendered or not) is rendered throught the template *`layout`*.
6. The document's content is modified.
7. Done!

4.3.7 Classes and Functions

`exception DocumentNotFound`

`class JinjaRenderer`(*config*)
This class holds the Jinja2 environment, removing the need to create it each time.

Attributes

env [`jinja2.Environment`] The configured Jinja environment.

layouts_patterns [str] The list of patterns which will be used to decide if the document should be processed.

layouts_default [str] The option so the `JinjaRenderer.render_content()` can access it.

`render_content`(*self*, *site*)

`exception RelativeLinkNotSupported`

`link`(*context*, *path*)

`setup`(*site*)

4.4 `taika.ext.markdown` – Markdown

This extension parses the content of the documents into HTML using CommonMarkdown specifications.

4.4.1 Trigger

This extension is subscribed to the "doc-post-read" event.

4.4.2 Frontmatter

None.

4.4.3 Process

1. Reads the suffix of *path* and if it matches, process the document.
2. Modifies the suffix of *url* path to ".html".
3. Process the content with *marko.convert* and replaces it.
4. Done!

4.4.4 Configuration

All configuration hangs from a key in the YML configuration named 'markdown'. Inside it, you can add the following options:

`suffixes`(*list*)
Default: **[.md]**

Tells the parser to ONLY modify docs with that suffix. Otherwise the document is ignored. This is checked against the source path (*path*), not the destination path (*url*).

4.4.5 Functions

`parse`(*site*, *document*)

`setup`(*site*)

4.5 `taika.ext.rst` – ReStructuredText

This extension parses the content of the documents into HTML using ReStructuredText specifications.

4.5.1 Trigger

This extension is subscribed to the "doc-post-read" event.

4.5.2 Frontmatter

None.

4.5.3 Process

1. Reads the suffix of *path* and if it matches, process the document.
2. Modifies the suffix of *url* path to ".html".
3. Process the content with `docutils.publish_parts()` and replaces it with the "body" part.
4. Done!

4.5.4 Configuration

Note: All configuration hangs from a key in the YML configuration named 'restructuredtext'. Inside it, you can add the following options:

`suffixes` (*list*)
Default: **[.rst]**

Tells the parser to ONLY modify docs with that suffix. Otherwise the document is ignored. This is checked against the source path (*path*), not the destination path (*url*).

`strict` (*bool*)
Default: **True**

Exits with error code 1 if there is any warning or error when parsing files.

`options` (*dict*)

Default:

{ stylesheet_path: '',
halt_level: 1,
traceback: True,
report_level: 5,
syntax_highlight: 'short',
doctitle_xform: False }

You can check the available options at HTML writer documentation

4.5.5 Functions

`parse_rst`(*site*, *document*)

Parse `content` and modify `url` keys of *document*.

Parameters

site [`taika.taika.Taika`] The Taika site.

document [dict] The document to be parsed.

`setup`(*site*)

CHAPTER 5

Internals

Take a look here if you want to help improve Taika or want to learn about how Taika is managed.

5.1 Contributing to Taika

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

You can contribute in many ways:

5.1.1 Types of Contributions

Fix Bugs

Look through the issue board for issues tagged as “type: bug”.

Implement Features

Look through the issue board for issues tagged as “type: feature-request”or “type: enhancement”.

Write Documentation

Documentation is always welcome, whether as part of the official Taika docs, in docstrings, or even on the web in blog posts, articles, and such. Don’t be shy and contribute :-D

Submit Feedback

The best way to send feedback is to file an issue at issue board. Select the template that fits your needs and submit it!

5.1.2 Get Started!

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

1. Fork the *taika* repo on GitLab.
2. Clone your fork locally:

```
$ git clone git@gitlab.com:your_name_here/taika.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 taika -p python3.6
$ cd taika/
$ python setup.py develop
$ pip install -r requirements.txt
$ pre-commit install
```

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, run tox:

```
$ tox
```

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

```
$ git add .
$ git commit
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a merge request through the GitLab website.

5.1.3 Merge Request Guidelines

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

1. Tests musts be included.
2. Documentation (docstrings or pages) must be included.
3. Python 3.6 should be supported.
4. The code should be isort'ed and flake8'ed. Optionally pylint'ed.

After you submit a merge request, check regularly the merge request, as Continuous Integration is run for each of your commits and should pass in order to merge the request.

5.1.4 Tips and tricks with pytest

Run all the test environments.

$ tox

To run only one environment:

```
$ tox -e <testenv>
```

To run a test module:

```
$ pytest tests/test_module
```

To run a test function inside a module:

```
$ pytest tests/test_module.py::test_function
```

CHAPTER 6

Reference

Take a look here if you want to use Taika's functions or classes yourself.

6.1 Document specification

Taika works with a dictionary representation of the documents. The following keys are defined in the documents when are read by Taika. The document can contain other keys, but they will be added by extensions:

`path` (*pathlib.Path*)

The path that the file has in the source folder. **Shouldn't be modified**.

`url` (*pathlib.Path*)

The path that the file will have in the destination folder. **Can be modified**.

`raw_content` (*bytes*)

The content that has the file in the source directory. **Shouldn't be modified**.

`content` (*bytes*)

The content after splitting the frontmatter from it. **Can be modified**.

6.2 Configuration file

Taika features configuration using YAML files, since their are readable and flexible. Here you will find the reference for the configuration file.

Warning: Taika reserves to words in the configuration: **extensions** and **extensions_path**. Overriding these two keys in the configuration file can lead to unexpected behaviour.

extensions (*list*)
A list of extensions to use.

E.g.:

```
extensions:
   - taika.ext.rst
   - taika.ext.permalinks
```

extensions_paths (*list*)
A list of paths where extensions live. This paths will be added to the `sys.path` in order to make the extensions inside it discoverable.

E.g.:

```
extensions_paths:
   - ./extensions
   - ./plugins
   - ./_extensions
   - /.extensions
   - ~/.extensions
```

6.3 taika

The top-level package contain some meta info about the package to be accessible by other tools.

__author__ (*str*)
The author name.

__email__ (*str*)
The email of *__author__*.

__version__ (*str*)
The version of the package.

class Taika (*source*, *destination*, *conf_path=None*)
Taika main class.

Attributes

source [`pathlib.Path`]

destination [`pathlib.Path`]

events [*`taika.events.EventManager`*]

config [dict]

documents [list]

import_extensions (*self*)
Load the configuration and extensions.

process (*self*)
Run *`Taika.read()`* and *`Taika.write()`*.

read (*self*, *source*)
Read all the files *recursively* from a *source* directory and load them as dictionaries.

Parameters

source [`pathlib.Path`] The source directory where the documents are read from.

Returns

documents [list] A list of dictionaries that represent documents.

`write`(*self*, *documents*, *destination*)

Call *taika.taika.write_file* for each document on *documents* with *destination*.

Parameters

documents [list] A list of dictionaries that represent documents.

destination [str or `pathlib.Path`] The destination directory.

6.4 `taika.cli`

The entry point for the command line interface of Taika.

`main`(*arguments=None*)

The main entry point, parse *arguments* behaves accordingly.

Parameters

arguments [list, optional (default=None)] A list of arguments to be parsed. If `None`, `sys.argv[1:]` is used.

Returns

err_code [int] Non-zero value indicates error, or zero on success.

`parse_arguments`(*arguments*)

Create a `argparse.ArgumentParser` and run `argparse.ArgumentParser.parse_args()` agains *arguments*.

Parameters

arguments [list] A list of arguments to be parsed.

Returns

namespace [*argparse.Namespace*] The namespace created when *arguments* are parsed.

6.5 `taika.events` – Basic event managment

This module offers a simple event manager implemented in the class *`taika.events.EventManager`*.

`events = {'doc-post-read', 'site-post-read'}`

The events that the event manager can register functions to.

`exception EventNotFound`

Exception raised when an event does not exists.

`class EventManager`

Register functions to events and passes them arguments and keyword arguments when called.

`call`(*self*, *event*, **args*, ***kwargs*)

Call all the functions registered to *event* passing **args* and **kwargs*.

Parameters

event [str] The event which be triggered.

Raises

EventNotFound If the *event* that is being triggered does not exist.

register(*self*, *event*, *func*)
Register a callable *func* to an *event*.

Parameters

event [str] The event to which *func* will be registered.

func [callable] A callable that will recieve arguments and keywords arguments when *event* is triggered.

Returns

current_id [int] The ID assigned to the function.

Raises

EventNotFound If the *event* that is being triggered does not exist.

6.6 taika.taika

read_conf(*conf_path*)
Read the configuration file *conf_path*. It should be an INI style configuration.

Parameters

conf_path [str] The path to the configuration file to be readed.

Returns

conf [*configparser.ConfigParser*] An instance of a ConfigParser which holds the configuration.

Raises

SystemExit If *conf_path* is not a file.

write_file(*document*, *destination*)
Given a *document* and a destionation, write *document.content* in the destination.

Parameters

document [dict] A dictionary representing a document. Should have `content` and `url`.

destination [str or `pathlib.Path`] The destination directory where the document will be written.

Raises

KeyError If the document doesn't have `content` or `url`.

read_file(*path*)
Read *path* and return the document as a dictionary.

Parameters

path [str or `pathlib.Path`] A path to a file to be read.

Returns

document [dict] A dictionary that holds the information of the document read from *path*.

CHAPTER 7

History

7.1 Unreleased

7.1.1 Added

- *taika.ext.markdown* extension to process markdown.

7.1.2 Changed

- Changed how *taika.ext.layouts* accept patterns to decide if a file should be rendered.
- Changed how *taika.ext.layouts* handle links: simplified function and only absolute links are supported.

7.2 v0.6.0 (2019-06-22)

7.2.1 Changed

- Changed how *taika.ext.collections* receive a pattern for including files

in a collection.

7.3 v0.5.3 (2019-06-13)

7.3.1 Fixed

- Test extension were links, which are not supported on Windows. Changed them to

plain files. * UnicodeError when installing on Windows due to name in pyproject.toml. Removed problematic characters. * Layouts extension link was returning a Path instead of an URL, so in Windows it was not working properly.

7.3.2 Removed

- Root Makefile, docs Makefile and docs make.bat. Now Tox uses another approach

to build documentation.

7.4 v0.5.2 (2019-05-11)

7.4.1 Fixed

- `taika.ext.rst` was printing `suffixes`.
- Now documents have *url* which is used to specify the path in the dest directory.
- Modified the extensions for the new document key *url*.
- rST include directory now works as it should.

7.4.2 Added

- `collections` extension that groups documents per pattern.
- `excerpt` extension added.
- `rst` extension now reads options from config file.

7.5 v0.5.1 (2018-04-16)

7.5.1 Changed

- The metadata was saying that the package was compatible with versions of Python and was wrong. Tags, classifiers and requires added.
- Files are read as bytes, so all the plugins and tests were adapted.

7.6 v0.5.0 (2018-04-16)

7.6.1 Added

- Extensions system.
- Two extensions: `rst` and `layouts`.
- INI file configuration.
- Main `Taika` class to orchestrate managers and configuration.
- `taika.ext.rst` now exits on warnings.

7.6.2 Changed

- CLI parsing now is done by `argparse`.

7.6.3 Fixed

- Documentation.

7.7 v0.4.0 (2018-03-17)

7.7.1 Added

- CLI entry point via `taika`.
- GitLab folder for issues and merge requests customization.
- Spell checker for the documentation.

7.7.2 Removed

- Certain folders that should be untracked.
- Unused badges on the README.

7.8 v0.3.0 (2018-03-15)

Necessary BUMP to wrap my head around the schema.

7.9 v0.2.1 (2018-03-15)

7.9.1 Added

- GitLab Continuous Integration.
- Configuration for pytest: now the working directory is the `tests` folder.

7.9.2 Removed

- Travis Continuous Integration.

7.10 v0.2.0 (2018-03-15)

7.10.1 Added

- Added the skeleton for the project.
- Added the first functions and functionality via API.

7.11 0.1.X (YYYY-MM-DD)

This versions correspond to older taika versions that I’ve uploaded to PyPi.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index