# Pysmith

**Dave Korhumel**

**Aug 21, 2020**

# CONTENTS

Welcome to the documentation site for Pysmith. Pysmith is a file processing pipeline heavily inspired by Metalsmith. It is primarily designed for static site generation, but is not limited to that. To use simply construct a `Pysmith` object, add the relevant plugins using `use()`, and execute the pipeline by calling `build()`.

The project's source can be found on Github here: https://github.com/dpk2442/pysmith.

CHAPTER

# ONE

# PYSMITH

**class** pysmith.**Pysmith**(*, *src*, *dest*)

The main pipeline processing object. To minimize potential issues, it is recommended that all paths are passed as absolute paths.

**Parameters**

- **src** (*str*) – The path to the source directory.
- **dest** (*str*) – The path to the destination directory.

**enable_logging**()

Enables basic logging while the build is running.

**Returns** self

**use**(*plugin*)

Add a new plugin instance to the pipeline.

**Parameters** **plugin** – A plugin that implements the *build()* method.

**Returns** self

**clean**()

Recursively deletes the destination directory.

**Returns** self

**build**()

Executes the plugins in the pipeline to run the build.

CHAPTER

TWO

# PLUGIN INTERFACE

## 2.1 Contract

Plugins are simply an object that implements the appropriate contract, which is a build method that follows the following signature:

**build**(*self*, *build_info*)
Executes the plugin in the current build.

**Parameters** **build_info** (BuildInfo) – The information for the current build.

## 2.2 Types

The following types are passed to plugins during the build.

**class** pysmith.**BuildInfo**
Contains information for the current build.

**metadata: dict(str, object)**
The global metadata for the build.

**files: dict(str, FileInfo)**
The files uses the file path relative to the source folder as the key and *FileInfo* objects as the values. Keys can be added or removed by the plugin, and files can be modified. At the end of the processing pipeline, files will be written based on the keys relative to the destination directory.

**get_files_by_pattern**(*match_pattern*)
Retrieves files from the *files* object using the given glob pattern.

**Parameters** **match_pattern** (*str*) – The pattern to pass to fnmatch.filter().

**get_files_by_regex**(*regex*)
Retrieves files from the *files* object using the given regex.

**Parameters** **regex** (re.Pattern) – The regular expression to use, compiled using re.compile().

**rename_file**(*file_name*, *new_file_name*)
Renames a file in the *files* object. If the new file name matches the old, no action is taken.

**Parameters**

- **file_name** (*str*) – The existing name of the file.
- **new_file_name** (*str*) – The new name for the file.

**class** pysmith.**FileInfo**
Contains the data for a file to be processed.

**name:** **str**
The file name.

**path:** **str**
The full path of the file.

**stats:** **os.stat_result**
The stats of the file, as returned by `os.stat()`.

**metadata:** **dict(str, object)**
The metadata of the file. This is a generic dictionary that can be modified by plugins.

**contents:** **bytes**
The raw binary contents of the file.

## 2.3 Logging

If plugins wish to log message, they should create a logger using `logging.getLogger()`, passing `pysmith.plugin.<plugin_name>` as the logger name. *enable_logging()* configures logging under the `pysmith` namespace, so plugins should use this name for logging to be configured correctly.

## 2.4 Utils

The following utilities are also available for plugins to use.

pysmith.plugin_util.**lambda_or_metadata_selector**(*selector*)
Ensures that the given selector is a function. If a str is provided, it will be converted into a function that queries the file's `metadata` for the provided key.

**Parameters** **selector** – The selector.

**Returns** func(*FileInfo*)

CHAPTER

THREE

# BUILT-IN PLUGINS

The following plugins come bundled with Pysmith. To automatically install the dependencies needed for each plugin, specify the extra specified in the plugin's documentation.

## 3.1 Core

The following plugins are core plugins and generally useful.

- *Collection*
  - This plugin creates a collection of files.
- *Frontmatter*
  - This plugin parses YAML frontmatter into the file's metadata.

## 3.2 Web

The following plugins are designed specifically for building static sites.

- *Markdown*
  - This plugin renders markdown files into HTML.
- *Minify*
  - This plugin minifies javascript using jsmin.
- *Permalink*
  - This plugin renames files so that the resulting URLs are pretty.
- *Sass*
  - This plugin compiles sass/scss files into css.
- *Template*
  - This plugin processes the contents of the file using a template.

### 3.2.1 Core

#### Collection

**class** pysmith.contrib.core.collection.**Collection**(*, *collection_name*, *match_pattern*, *order_by*, *reverse=False*)

Creates ordered collections of files. Collections are sorted lists of *FileInfo* objects that are stored in a dictionary (using the provided name as a key) in the build info *metadata* under the key *collections*.

**Parameters**

- **collection_name** (*str*) – The name of the collection.
- **match_pattern** (*str*) – The pattern of files to build the collection from.
- **order_by** (str or func(*FileInfo*)) – The function to use when sorting the collection. If this is a string, it will be used as the key to look up the value to order by in the file's *metadata*. If this is a function, it will be executed to find the value to order by.
- **reverse** (*bool*) – If this parameter is true, the collection will be sorted in reverse order.

#### Frontmatter

The dependencies necessary for this plugin can be installed using the `frontmatter` extra.

**class** pysmith.contrib.core.frontmatter.**Frontmatter**(*, *match_pattern='*'*)

Parses YAML frontmatter from files. The parsed frontmatter metadata will be added to the file's *metadata* and removed from the *contents*.

**Parameters** **match_pattern** (*str*) – The pattern of files to parse metadata from.

### 3.2.2 Web

#### Markdown

The dependencies necessary for this plugin can be installed using the `markdown` extra.

**class** pysmith.contrib.web.markdown.**Markdown**(*, *match_pattern='*.md'*, *extras=None*)

Renders markdown into html. The file's *contents* will be updated and the source file will not be renamed.

**Parameters**

- **match_pattern** (*str*) – The pattern of files to render.
- **extras** (*list(str) or None*) – A list of extras to pass to markdown2.

#### Minify

The dependencies necessary for this plugin can be installed using the `minify` extra.

**class** pysmith.contrib.web.minify.**Minify**(*js_match_pattern='*.js'*)

Minifies javascript using `rjsmin.jsmin()`. The file's *contents* will be updated and the source file will not be renamed.

**Parameters** **js_match_pattern** (*str*) – The pattern of javascript files to minify.

### Permalink

**class** pysmith.contrib.web.permalink.**Permalink**(*match_pattern='*.html'*, *permalink_selector='permalink'*)

Creates permalinks for pages. The permalink will be pulled from the *FileInfo* object as specified by the *permalink_selector*, and the file will be moved under the correct key for the new path. If the permalink cannot be found in the *FileInfo*, the file will not be renamed. Leading slashes are ignored in the permalink. If the path ends with a slash, the file will be moved to *index.html* in the specified folder. Otherwise the permalink will be treated as a literal path for the file.

**Parameters**

- **match_pattern** (*str*) – The pattern of files to rename.
- **permalink_selector** (str or func(*FileInfo*)) – The permalink selector. If this is a string, it will be used as the key to look up the permalink in the file metadata. If this is a function, it will be executed to find the permalink.

### Sass

The dependencies necessary for this plugin can be installed using the `sass` extra.

**class** pysmith.contrib.web.sass.**Sass**(*, *match_pattern='.*\\.(sass|scss)'*, *output_extension='.css'*, *compile_args={}*)

Compiles sass/scss into css. The file's *contents* will be updated and the file will be renamed if necessary based on the output extension.

**Parameters**

- **match_pattern** (*str*) – A regex pattern to specify which files to compile.
- **output_extension** (*str*) – The extension the file should have after compilation.
- **compile_args** (*dict(str, object)*) – Extra arguments to be passed to `sass.compile()` in addition to the file contents.

### Template

The template plugin is actually two plugins. The first treats the content of the file itself as the template, whereas the second inserts the content into a layout template. This plugin uses jinja2 to do all templating. The dependencies necessary for this plugin can be installed using the `template` extra.

**class** pysmith.contrib.web.template.**_BaseTemplate**(*, *match_pattern='*'*, *globals=None*, *global_include=None*, *environment_args={}*)

This class is not intended to be instantiated directly, but instead just serves to hold common logic for both template plugins.

**Parameters**

- **match_pattern** (*str*) – The pattern of files to process.
- **globals** (*dict(str, object)*) – Global values to insert into the underlying `Environment`.
- **global_include** (*str*) – The name of a template (to be retrieved using `get_template()`) containing macros that will be included in the globals list of the `Environment`. The macros will be available in all templates, but will not have access to the rendering context.

- **environment_args** (*dict(str, object)*) – A dictionary of options to pass to the `Environment` constructor.

**class** pysmith.contrib.web.template.**ContentTemplate**(*, **kwargs*)

Treats the contents of the files as a template and renders it. This can be used to do pre-processing on the source files. All parameters specified in *_BaseTemplate* are valid for this class as well. When the template is rendered, the build info *metadata* will be available as `site`.

**class** pysmith.contrib.web.template.**LayoutTemplate**(*, *layout_selector='layout'*, *output_extension='.html'*, **kwargs*)

Treats the contents of the files as a variable to pass into a template. The layout to use is selected by the `layout_selector` parameter. When the template is rendered, the file *contents* will be available in the rendering context as `contents`, the file *metadata* will be available as `page`, and the build info *metadata* will be available as `site`. In addition to the parameters specified below, all parameters specified in *_BaseTemplate* are valid for this class as well.

**Parameters**

- **layout_selector** (str or func(*FileInfo*)) – The layout selector. If this is a string, it will be used as the key to look up the template in the file metadata. If this is a function, it will be executed to find the name of the template to use.
- **output_extension** (*str*) – The extension to use for the output file. The file info will be moved to a new key in the files dictionary if the file needs to be renamed to have the correct extension.

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- search

# PYTHON MODULE INDEX

## p

# INDEX

## Symbols



## B



## C



## E



## F



## G



## L



## M



## N



## P



## R



## S



## U