Graaf Documentation

Release 0.3.2

Curtis Maloney

Mar 13, 2017

Contents:

1 Config file format **1**

2 graaf package **3**
- 2.1 Submodules . . . 3
- 2.2 graaf.base module . . . 3
- 2.3 graaf.scss module . . . 4
- 2.4 graaf.simple_md module . . . 4
- 2.5 graaf.verbatim module . . . 4
- 2.6 Module contents . . . 4

3 Why Graaf? **5**

4 Why did you build it? **7**

5 How it works **9**

6 Configuration **11**
- 6.1 paths . . . 11
- 6.2 generators . . . 11

Python Module Index **13**

CHAPTER 1

Config file format

By default, Graaf will read *graaf.yml* for settings.

The layout is as follows, with default values provided:

```
paths:
  source: pages/
  dest: assets/
  templates: templates/

generators:
  - name: graaf.simple_md.SimpleMarkdown
  - name: graaf.scss.SassGenerator
```

The *generators* section lists configurations for the Generators to use, in order. Any other values beyond 'name' will be passed to the class when it's instantiated.

CHAPTER 2

graaf package

Submodules

graaf.base module

class `graaf.base.`**`Generator`**

Bases: `object`

Base Generator class

`can_process`(*filename*)

Determine if this instance will attempt to process the file.

`enter_dir`(*src_dir*, *dirs*, *files*)

Hook to allow Generators to react to start of processing a dir.

`extensions` = []

`finish`(*processor*)

Hook to allow Generators to react to end of processing.

`leave_dir`(*src_dir*, *dirs*, *files*)

Hook to allow Generators to react to end of processing a dir.

`process`(*src_dir*, *dest_dir*, *filename*, *processor*)

Called by the Processor for any file that can_process returns True for.

`read_file`(*src_dir*, *filename*)

`start`(*processor*)

Hook to alow Generators to react to start of processing.

`write_file`(*dest_dir*, *filename*, *content*)

class `graaf.base.`**`Processor`**(*srcdir*, *destdir*, *templatedir*, *generators=None*)

Bases: `object`

Site Processor class.

Given directories for source documents, destination, and templates, as well as a list of generators, will process all files.

Will apply *_.yml* to the context in each directory.

render(*template_name*, *extra_context*)
Helper function to render a template with the current context, and extra context.

run()
Trigger to process...

graaf.base.**get_yaml**(*src*)
Try to read a YAML document from the provided file.

If it doesn't exist, return an empty dict instead.

graaf.scss module

graaf.simple_md module

class graaf.simple_md.**SimpleMarkdown**
Bases: *graaf.base.Generator*

Generator for processing plain Markdown files

Will try to read {filename}.yml for additional context. Processes content as a Template first, allowing variables, etc. Will use context['template'] for the template name.

extensions = ['.md']

process(*src_dir*, *dest_dir*, *filename*, *processor*)

graaf.verbatim module

class graaf.verbatim.**VerbatimGenerator**
Bases: *graaf.base.Generator*

Copy _any_ file verbatim to the destination.

Useful for static content like images and fonts.

can_process(*filename*)

process(*src_dir*, *dest_dir*, *filename*, *processor*)

Module contents

graaf.**get_version**()

CHAPTER 3

Why Graaf?

This project is named after Van de Graaf.

CHAPTER 4

Why did you build it?

Recently I've built a number of sites backed by AWS API Gateway, and needed a way to build their static pages, but remain consistent with the templates and styles of the rest of the site.

CHAPTER 5

How it works

1. Put your source content in a directory. By default this is called *pages*.
2. You create a target directory to generate the content in. By default this is called *assets*.
3. Write templates for your content. By default these go in a directory called *templates*.
4. You run *graaf*

   It will find every file in your sirce directory, and check if any of its configured Generators will work on it. If so, they will be invoked, and (typically) generate an output file.
5. PROFIT!

CHAPTER 6

Configuration

The main configuration file is called `graaf.yml` and contains two main sections:

paths

Here you can override any of three directories

```
paths:
  srcdir:
  destdir:
  templates:
```

generators

This is a list of generator classes to use, in order.

Each item in the list is a map with at least the import name of a Generator class. It may also include arguments to pass to the Generator for configuration.

```
generators:
  - name: 'graaf.simple_md.SimpleMarkdown'
  - name: 'graaf.scss.SassGenerator'
```

Python Module Index

g

graaf, 4
graaf.base, 3
graaf.simple_md, 4
graaf.verbatim, 4

Index

C

can_process() (graaf.base.Generator method), 3
can_process() (graaf.verbatim.VerbatimGenerator method), 4

E

enter_dir() (graaf.base.Generator method), 3
extensions (graaf.base.Generator attribute), 3
extensions (graaf.simple_md.SimpleMarkdown attribute), 4

F

finish() (graaf.base.Generator method), 3

G

Generator (class in graaf.base), 3
get_version() (in module graaf), 4
get_yaml() (in module graaf.base), 4
graaf (module), 4
graaf.base (module), 3
graaf.simple_md (module), 4
graaf.verbatim (module), 4

L

leave_dir() (graaf.base.Generator method), 3

P

process() (graaf.base.Generator method), 3
process() (graaf.simple_md.SimpleMarkdown method), 4
process() (graaf.verbatim.VerbatimGenerator method), 4
Processor (class in graaf.base), 3

R

read_file() (graaf.base.Generator method), 3
render() (graaf.base.Processor method), 4
run() (graaf.base.Processor method), 4

S

SimpleMarkdown (class in graaf.simple_md), 4
start() (graaf.base.Generator method), 3

V

VerbatimGenerator (class in graaf.verbatim), 4

W

write_file() (graaf.base.Generator method), 3