Fabricius

Release 0.1.0.dev1

Predeactor

Dec 03, 2022

GUIDES:

Fabricius: Python modular render of template engine & project scaffolding.

CHAPTER

ONE

EXAMPLE

```
from fabricius import Generator

def create_files():
    # Create the generator
    generator = Generator()

    # Add a file at "source/core.py"
    file = generator.add_file("core", "py")
    file.from_file("template-core.mustache").to_directory("source").with_data({"name":
↪"My Module"}).use_mustache()

    # Add a file at "tests/test.py"
    file = generator.add_file("test", "py")
    file.from_file("template-test.txt").to_directory("tests").with_data({"name": "My␣
↪Module"})

    # Create the files!
    generator.execute()
```

CHAPTER

TWO

LINKS

2.1 Guide: Rendering programmatically

Fabricius was based on the idea to be able to programmatically create your files. As such, it uses Python code to tell how a file should be renderer, where to go, from what template, etc. You can manually create a *FileGenerator* object, or you can also uses *Generator* to create multiple files at one time and get a feedback on your terminal.

Let's create a simple template that can be renderer by Python's `str.format`:

Listing 1: template/source.txt

```
Hello! I am template!
Let's render the date of today: {date}

I use {renderer} for rendering!
```

Now that we have our template, we need to *actually* render the file.

Listing 2: source/conf.py

```
from fabricius import FileGenerator
from datetime import datetime

def run_me():
    file = FileGenerator("my_file", "txt")

    # Define the template file - You can also use .from_content
    file.from_file("template/source.txt")

    # Define the destination
    file.to_directory("destination/")

    # Include the data to pass for rendering
    file.with_data({date: datetime.now(), renderer: "Python's .format"})

    # And finally, save the file!
    file.commit()

    # If you wish to simply generate the file's content without saving it, you can use .
↪generate
    final_content = file.generate()
```

(continues on next page)

(continued from previous page)

```
run_me()
```

2.2 fabricius package

2.2.1 Subpackages

fabricius.generator package

Submodules

fabricius.generator.errors module

exception fabricius.generator.errors.**NoContentError**(*file_name: str*)

Bases: FabriciusError

The file does not have any content.

exception fabricius.generator.errors.**NoDestinationError**(*file_name: str*)

Bases: FabriciusError

The file does not know where to go.

exception fabricius.generator.errors.**AlreadyCommittedError**(*file_name: str*)

Bases: FabriciusError

The file has already been committed/persisted.

fabricius.generator.file module

class fabricius.generator.file.**GeneratorCommitResult**

Bases: TypedDict

A CommitResult is returned when a file was successfully saved. It returns its information after its creation.

name: str

The name of the file.

state: Literal['pending', 'persisted', 'deleted']

The state of the file. Should always be "persisted".

destination: Path

Where the file is located/has been saved.

data: Dict[str, Any]

The data that has been passed during rendering.

template_content: str

The original content of the template.

content: str

The resulting content of the saved file.

class fabricius.generator.file.**FileGenerator**(*name: str*, *extension: Optional[str] = None*)

Bases: object

template_content: Optional[str]

The content of the base template, if set.

name: str

The name of the file that will be generated.

state: Literal['pending', 'persisted', 'deleted']

The state of the file.

destination: Optional[Path]

The destination of the file, if set.

renderer: Type[*Renderer*]

The renderer to use to generate the file.

data: Dict[str, Any]

The data that will be passed to the renderer.

from_file(*path: Union[str, PathLike[str], Path]*) → Self

Read the content from a file template.

Raises

FileNotFoundError – If the file was not found.

Parameters

path (str or pathlib.Path) – The path of the file template.

from_content(*content: str*) → Self

Read the content from a string.

Parameters

content (str) – The template you want to format.

to_directory(*directory: Union[str, PathLike[str], Path]*) → Self

Set the directory where the file will be saved.

Raises

NotADirectory – The given path exists but is not a directory.

Parameters

directory (str or pathlib.Path) – Where the file will be stored. Does not include the file's name.

use_mustache() → Self

Use chevron (Mustache) to render the template.

use_string_template() → Self

Use string.Template to render the template.

with_renderer(*renderer: Type[*Renderer*]*) → Self

Use a custom renderer to render the template.

Parameters

renderer (Type of *fabricius.generator.renderer.Renderer*) – The renderer to use to format the file. It must be not initialized.

with_data(*data: Dict[str, Any]*, ***, *overwrite: bool = True*) → Self

Add data to pass to the template.

Parameters

- **data** (*fabricius.const.Data*) – The data you want to pass to the template.
- **overwrite** (bool) – If the data that already exists should be deleted. If False, the new data will be added on top of the already existing data. Default to `True`.

generate() → str

Generate the file's content.

Raises

fabricius.generator.errors.NoContentError – If no content to the file were added.

Returns

The final content of the file.

Return type

str

commit(***, *overwrite: bool = False*, *dry_run: bool = False*) → *GeneratorCommitResult*

Save the file to the disk.

Parameters

- **overwrite** (bool) – If a file exist at the given path, shall the overwrite parameter say if the file should be overwritten or not. Default to `False`.
- **dry_run** (bool) – You should not use this. This is mostly used for Fabricius's tests. This parameter indicate if files should be created.

Raises

- *fabricius.generator.errors.NoContentError* – If no content to the file were added.
- *fabricius.generator.errors.NoDestinationError* – If no destination/path were designated.
- *fabricius.generator.errors.AlreadyCommittedError* – If the file has already been saved to the disk.
- **FileExistsError** – If the file already exists on the disk and `overwrite` is set to `False`.

  This is different than *AlreadyCommittedError* because this indicate that the content of the file this generator was never actually saved.

Returns

A typed dict with information about the created file.

Return type

`fabricius.generator.file.CommitResult`

fabricius.generator.generator module

class fabricius.generator.generator.**Generator**

Bases: SupportsPlugin[*GeneratorPlugin*]

files: List[*FileGenerator*]

The list of files to generate with the generator.

add_file(*name: str*, *extension: Optional[str] = None*) → *FileGenerator*

Add a file to the generator.

Parameters

- **name** (str) – The name of the file
- **extension** (Optional, str) – The extension of the file, can be optional. If none, no extension will be added.

Returns

The generated file. You then have to set file's options.

Return type

fabricius.generator.file.FileGenerator

execute(*, *allow_overwrite: bool = False*, *dry_run: bool = False*) → Dict[*FileGenerator*, Optional[*GeneratorCommitResult*]]

Execute generator's tasks.

Parameters

- **allow_overwrite** (bool) – If files exist at their set path, shall this parameter say if files should be overwritten or not.
- **dry_run** (bool) – You should not use this. This is mostly used for Fabricius's tests. This parameter indicate if files should be created.

Returns

A dict containing a file generator and its commit result. In case the value is None, this mean that the file was not successfully saved to the disk (Already committed, file already exists, etc.).

Return type

Dict[*fabricius.generator.file.FileGenerator*, fabricius.generator.file.CommitResult]

fabricius.generator.renderer module

class fabricius.generator.renderer.**Renderer**(*data: Dict[str, Any]*)

Bases: ABC

data: Dict[str, Any]

A dictionary that contains data passed by the users to pass inside the template.

abstract render(*content: str*) → str

class fabricius.generator.renderer.**PythonFormatRenderer**(*data: Dict[str, Any]*)

Bases: *Renderer*

render(*content: str*) → str

class fabricius.generator.renderer.**ChevronRenderer**(*data: Dict[str, Any]*)

Bases: *Renderer*

render(*content: str*) → str

class fabricius.generator.renderer.**StringTemplateRenderer**(*data: Dict[str, Any]*)

Bases: *Renderer*

render(*content: str*) → str

Module contents

fabricius.plugins package

Submodules

fabricius.plugins.generator module

class fabricius.plugins.generator.**GeneratorPlugin**

Bases: BasePlugin

A plugin to plug to the *fabricius.generator.generator.Generator* class.

You can edit the methods of the class, and they'll be run according to their description.

on_file_add(*file:* FileGenerator) → Any

Called when a new file has been added to the generator.

Parameters
file (*fabricius.generator.file.FileGenerator*) – The file that has been added to the generator.

before_execution() → Any

Called when the user has called the "execute" method. This is ran before the generator creates any files.

before_file_commit(*file:* FileGenerator) → Any

Called when a file is about to be created. The file is NOT yet created and is still not saved locally.

Parameters
file (*fabricius.generator.file.FileGenerator*) – The file that will be generated.

after_file_commit(*file:* FileGenerator) → Any

Called when a file has been created and saved locally.

Parameters
file (*fabricius.generator.file.FileGenerator*) – The file that has been generated.

after_execution(*results: Dict[*FileGenerator, *Optional[*GeneratorCommitResult*]]*) → Any

Called when the generator has realized all file generation.

Parameters
results (List of *fabricius.generator.file.GeneratorCommitResult*) – A list of *GeneratorCommitResult*.

on_commit_fail(*file:* FileGenerator, *exception:* *Exception*) → Any

Called when the generator has failed to commit a file.

Parameters

- **file** (*fabricius.generator.file.FileGenerator*) – The file that has been generated.
- **exception** (Exception) – The exception that was raised.

Module contents

2.2.2 Submodules

2.2.3 fabricius.const module

fabricius.const.**Data**

Represent the data passed to a generator/file, under a form of dictionary.

alias of Dict[str, Any]

fabricius.const.**FILE_STATE**

Define the state of a file.

The file's state can be one of:

- pending
- persisted
- deleted

alias of Literal['pending', 'persisted', 'deleted']

fabricius.const.**PathStrOrPath**

Represent a path as a str or a pathlib.Path object. Typically what's used to treat path in Fabricius.

alias of Union[str, PathLike[str], Path]

2.2.4 fabricius.utils module

Utilities shipped with Fabricius.

fabricius.utils.**camel_case**(*text:* *str*) → str

Return the text formatted in camel case

Parameters
text (*str*) – The text you want to format.

Returns
The formatted text.

Return type
str

Example

```
>>> my_text = "Some text"
>>> camel_case(my_text)
'someText'
```

fabricius.utils.**snake_case**(*text: str*) → str

Return the text formatted in snake case

Parameters
text (*str*) – The text you want to format.

Returns
The formatted text.

Return type
str

Example

```
>>> my_text = "Some text"
>>> snake_case(my_text)
'some_text'
```

fabricius.utils.**dash_case**(*text: str*) → str

Return the text formatted in dash case

Parameters
text (*str*) – The text you want to format.

Returns
The formatted text.

Return type
str

Example

```
>>> my_text = "Some text"
>>> dash_case(my_text)
'some-text'
```

fabricius.utils.**pascal_case**(*text: str*) → str

Return the text formatted in pascal case

Parameters
text (*str*) – The text you want to format.

Returns
The formatted text.

Return type
str

Example

```
>>> my_text = "Some text"
>>> pascal_case(my_text)
'SomeText'
```

fabricius.utils.**capital_case**(*text: str*) → str

Return the text formatted in capital case

Parameters
text (*str*) – The text you want to format.

Returns
The formatted text.

Return type
str

Example

```
>>> my_text = "Some text"
>>> capital_case(my_text)
'Some Text'
```

fabricius.utils.**sentence_case**(*text: str*) → str

Return the text formatted in sentence case

Parameters
text (*str*) – The text you want to format.

Returns
The formatted text.

Return type
str

Example

```
>>> my_text = "Some text"
>>> sentence_case(my_text)
'Some text'
```

2.2.5 Module contents

2.3 fabricius

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

M

N

O

P

R

S

T

U

W