cookiecutter-python-package

Release 2.0.2

Konstantinos Lampridis

Feb 27, 2024

CONTENTS:

CHAPTER

ONE

PYTHON PACKAGE GENERATOR

Generate Python Project and enjoy **streamlined DevOps** using a powerful **CI/CD Pipeline.**

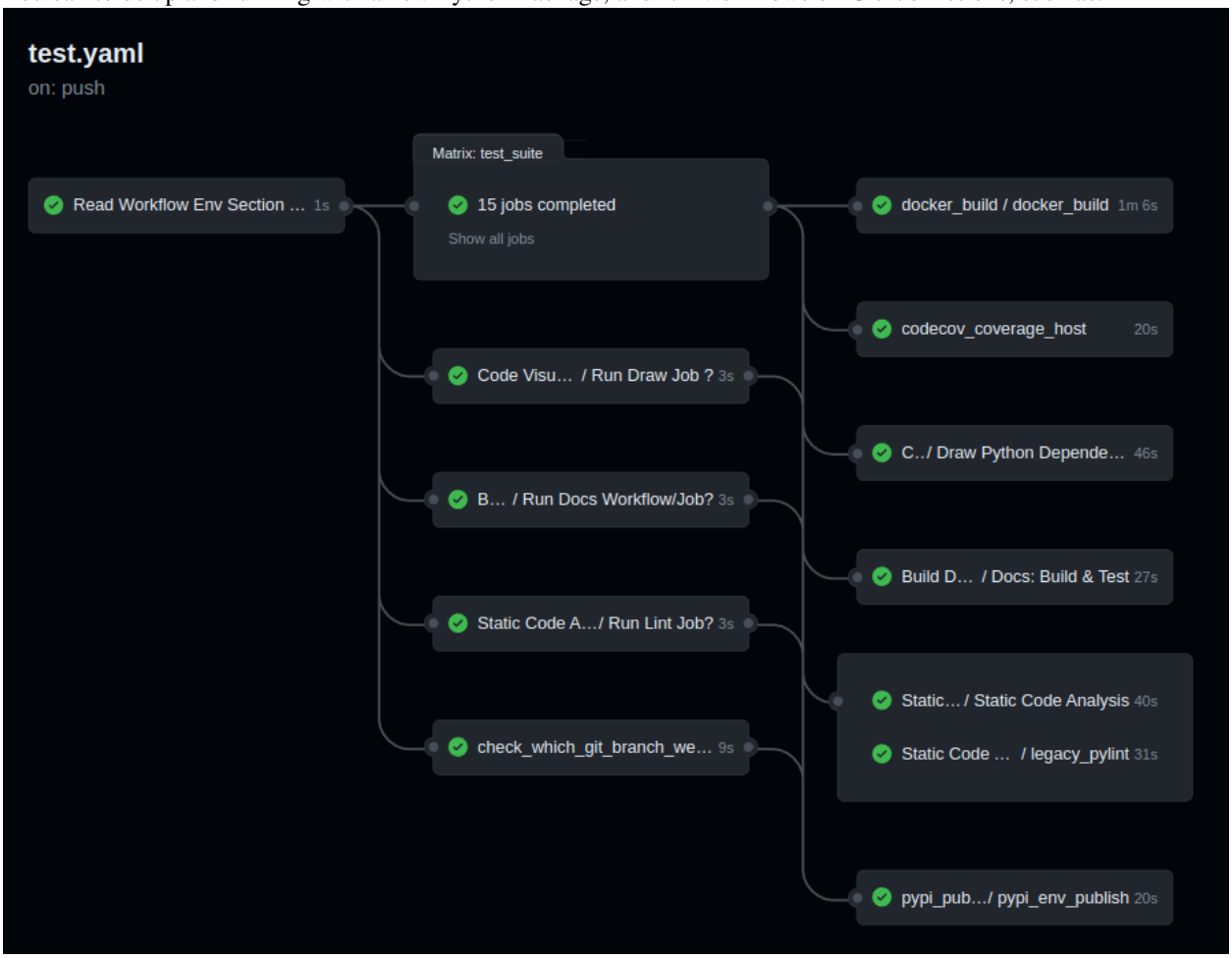
Documentation available at https://python-package-generator.readthedocs.io/.

1.1 What's included?

- **Generator** of **Python Project** (see *Quickstart*), with **CLI** for **Linux**, **MacOS**, and **Windows**
- **Option** to Generate Python Package designed as *module*, *module+cli*, or *pytest-plugin*!
- Scaffold over **24 files**, from Template, to have a *ready-to-develop* **Project equipped** with:
  - Fully-featured **CI/CD Pipeline**, running on Github Actions, defined in *.github/*
  - **Continuous Delivery** to *PyPI* (i.e. pypi.org, test.pypi.org) and *Dockerhub*
  - **Continuous Integration**, with **Test Suite** running pytest, located in the *tests* dir
  - **Continuous Documentation**, building with *mkdocs* or *sphinx*, and hosting on *readthedocs*, located in the *docs* dir
  - **Static Type Checking**, using mypy
  - **Lint** *Check* and *Apply* commands, using the fast Ruff linter, along with standard isort, and black
  - **Build Command**, using the build python package

1.2 What to expect?

You can to be up and running with a new Python Package, and run workflows on Github Actions, such as:



Link: https://github.com/boromir674/biskotaki/actions/runs/4157571651

1. **CI Pipeline**, running on Github Actions, defined in *.github/*
    a. **Job Matrix**, spanning different *platform*'s and *python version*'s
        1. Platforms: *ubuntu-latest*, *macos-latest*
        2. Python Interpreters: *3.6*, *3.7*, *3.8*, *3.9*, *3.10*
    b. **Parallel Job** execution, generated from the *matrix*, that runs the *Test Suite*
    c. **Artifact** store of **Source** and **Wheel** Distributions, factoring Platform and Python Version

1.2.1 Auto Generated Sample Package Biskotaki

Check the **Biskotaki** *Python Package Project*, for a taste of the project structure and capabilities this Template can generate!
It it entirely generated using this **Python Package Template:**

Source Code hosted on *Github* at https://github.com/boromir674/biskotaki
Python Package hosted on *pypi.org* at https://pypi.org/project/biskotaki/
CI Pipeline hosted on *Github Actions* at https://github.com/boromir674/biskotaki/actions

1.3 Quickstart

To **install** the latest `Generator` in your environment, run:

```
pip install cookiecutter-python
```

The `generate-python` (executable) CLI should now be available in your environment.
Next, **create** a file, let's call it `gen-config.yml`, with the following content:

```
default_context:
    project_name: Demo Generated Project
    project_type: module+cli
    full_name: John Doe
    email: john.doe@something.org
    github_username: john-doe
    project_short_description: 'Demo Generated Project Description'
    initialize_git_repo: no
    interpreters: {"supported-interpreters": ["3.8", "3.9", "3.10", "3.11"]}
```

To **generate** a Python Package Project, run:

```
mkdir gen-demo-dir
cd gen-demo-dir

generate-python --config-file ../gen-config.yml --no-input
```

Now, you should have generated a new Project for a Python Package, based on the Template!
The Project should be located in the newly created `demo-generated-project` directory.
To leverage all out-of-the-box development operations (ie scripts), install tox:

```
python3 -m pip install --user 'tox<4'
```

To verify tox available in your environment, run: `tox --version`
Please, do a *cd* into the newly created directory, ie *cd <my-great-python-package>*.
To run the Test Suite, *cd* into the newly created Project folder, and run:

```
tox -e dev
```

All Tests should pass, and you should see a *coverage* report!
To run Type Checking against the Source Code, run:

```
tox -e type
```

All Type Checks should pass!
To setup a Git Repository, run:

```
git init
git add .
git checkout -b main
git commit -m "Initial commit"
```

To setup a Remote Repository, run for example:

```
git remote add origin <remote-repository-url>
git push -u origin main
```

To trigger the CI/CD Pipeline, run:

```
git push
```

Navigate to your github.com/username/your-repo/actions page, to see the CI Pipeline running!

Develop your package's **Source Code** (*business logic*) inside *src/my_great_python_package* dir :)
Develop your package's **Test Suite** (ie *unit-tests*, *integration tests*) inside *tests* dir :-)

Read the Documentation's Use Cases section for more on how to leverage your generated Python Package features.

1.3.1 Next Steps

To prepare for an Open Source Project Development Lifecycle, you should visit the following websites:

- PyPI, test.pypi.org, Dockerhub, and Read the Docs, for setting up Release and Documentation Pipelines
- github.com/your-account to configure Actions, through the web UI
- Codecov, Codacy, and Codeclimate, for setting up Automated Code Quality, with CI Pipelines
- https://www.bestpractices.dev/ for registering your Project for OpenSSF Best Practices Badge

Happy Developing!

1.4 License

- GNU Affero General Public License v3.0

1.4.1 Free/Libre and Open Source Software (FLOSS)

Introduction

This is **Cookiecutter Python Package**, a *Template Project* used to *generate* fresh new open source *Python Package*'s.
The Template is implemented as a *cookiecutter* and it is available both as source code and as a Python Package in itself.

Goal of this project is to automate the process of creating a new Python Package, by providing the user with a "bootstrap" method,
to quickly set up all the *support* files required to seemlessly build and publish the package on pypi.org (the official Python Pcakge Index public server).
Additionally, it instruments a basic **Test Suite**, multiple **Commands**, as well as a **CI** pipeline, with parallel execution of the *build matrix*, running on *Github Actions*.

This documentation aims to help people understand what are the features of the library and how they can use it. It presents some use cases and an overview of the library capabilities and overall design.

Why this Generator?

So, why choose this Python Package Generator?

Robust CLI

You want an *easy-to-use*, *cross-platform* CLI.

- It offers an **1-click** command, or option for an interactive *wizard*
- **Tested** on **15 different setups**, across multiple *Platforms* and *Python Interpreters*
  - **OS**: {Ubuntu, MacOS, Windows} X **Python**: {3.7, 3.8, 3.9, 3.10, 3.11}
- Built on established software, such as *cookiecutter* and *jinja2*

"DevOps": aka Automations and CI/CD

You want to focus on your *business logic* and *test cases*, in new Python projects.

- Scaffolded project is **one push** away from triggering its **CI/CD pipeline** on Github Actions.
- **Continuous Deployment**, publishing at *pypi.org*, *Docker Hub*, and *Read The Docs*
- Designed for **GitOps**, supporting various *automated developer activities*
- **Automations** with same entrypoint for both **CI and Local** run, via *tox*
- Stress-Testing, with **Job Matrix** spanning multiple *Python Interpreters*, *Operating Systems*

Approved Tooling

You want the best tools under your belt, for your development lifecycle.

- *tox*, *poetry*, *ruff*, *mypy*, *pytest*, *black*, *isort*, *mkdocs*, *sphinx*

Template Variant

You want *poetry*, but what if you want to develop a *pytest plugin*?

- Generate *module*: a Python Distribution, with python API/sdk
  - configured with *poetry* backend
- Generate *module+cli*: a Python Distribution, with a CLI and a python API/sdk
  - configured with *poetry* backend
- Generate *pytest-plugin*: a Python Distribution, designed for a *pytest plugin*
  - configured with *setuptools* backend, as Required by *pytest*!

Generate Python Project

Our `Python Generator` **Project** was designed to be installable via *pip* and then invoked through the `generate-python` entrypoint to the CLI.

Getting the CLI

Cookiecutter Python Package, available as *source code* on github, with published **Distribution** on *pypi.org* **PyPI**, and **Docker Image** on *hub.docker.com* **Registry**.

Install in virtual env, and make available globally in your (host) machine.

```
pipx install cookiecutter-python
```

Now, the `generate-python` executable should be available.
Pull the latest Stable image from Docker Hub

```
docker pull boromir674/generate-python:master
```

Now, the CLI should be available, via
`docker run -it --rm boromir674/generate-python:master`

Hint: Not to be confused with the 'latest' (channel) image tag advertised by dockerhub, which my no means promises to contain a stable release.

Hint: We promise stable releases on **'master'** tag, since on git, 'all tagged production releases are on 'master' branch. They are the same to correspond to PyPI Distributions as well.

Install in virtual env

```
virtualenv env --python=python3
source env/bin/activate

pip install cookiecutter-python
```

Make available to current user

```
ln -s env/bin/generate-python ~/.local/bin/generate-python
```

Now, the `generate-python` executable should be available (assuming ~/.local/bin is in your PATH).

Check installation

You can verify by running the following:

```
generate-python --version
```

Hint: All methods demonstrated for **getting the CLI**, *download* the **Latest Stable** (Release).

Using the CLI

Using the cli is as simple as invoking *generate-python* from a console.

You can run the following to see all the available parameters you can control:

```
generate-python --help
```

```
docker run -it --rm boromir674/generate-python:master --help
```

The most common way to generate a new Python Package Project is to run:

```
generate-python
```

```
docker run -it --rm boromir674/generate-python:master
```

This will prompt you to input some values and create a fresh new Project in the current directory!
Now, simply *cd* into the generated Project's directory and enjoy some of the features the generator supplies new projects with!
More on use cases in the next section.

Generated Python Project Use Cases

Ready to enjoy some of your newly generated Python Package Project **features** available out-of-the-box!?
For instance:

1. Leverage the supplied *tox environments* to automate various **Testing** and **DevOps** related activities.
   Assuming you have *tox* installed (example installation command: *python3 -m pip install –user tox*) and you have done a *cd* into the newly generated Project directory, you can do for example:
   a. Run the **Test Suite** against different combinations of *Python versions* (ie 3.7, 3.8) and different ways of installing (ie 'dev', 'sdist', 'wheel') the <*my_great_python_package*> package:

      ```
      tox -e "py{3.7, 3.8}-{dev, sdist, wheel}"
      ```

   b. Check the code for **compliance** with **best practises** of the *Python packaging ecosystem* (ie PyPI, pip), build *sdist* and *wheel* distributions and store them in the *dist* directory:

      ```
      tox -e check && tox -e build
      ```

   c. **Deploy** the package's distributions in a *pypi* (index) server:
      1. Deploy to **staging**, using the *test* pypi (index) server at test.pypi.org:

         ```
         TWINE_USERNAME=username TWINE_PASSWORD=password PACKAGE_DIST_VERSION=1.0.0␣
         ↪tox -e deploy
         ```

      2. Deploy to **production**, using the *production* pypi (index) server at pypi.org:

         ```
         TWINE_USERNAME=username TWINE_PASSWORD=password PACKAGE_DIST_VERSION=1.0.0␣
         ↪PYPI_SERVER=pypi tox -e deploy
         ```

         Note: Setting PYPI_SERVER=pypi indicates to deploy to *pypi.org* (instead of *test.pypi.org*).

   Note: Please modify the TWINE_USERNAME, TWINE_PASSWORD and PACKAGE_DIST_VERSION environment variables, accordingly.
   TWINE_USERNAME & TWINE_PASSWORD are used to authenticate (user credentials) with the targeted pypi server.

PACKAGE_DIST_VERSION is used to avoid accidentally uploading distributions of different versions than intended.

2. Leverage the **CI Pipeline** and its **build matrix** to run the **Test Suite** against a combination of different Platforms, different Python interpreter versions and different ways of installing the subject Python Package:
   Trigger the **Test Workflow** on the **CI server**, by *pushing* a git commit to a remote branch (ie *master* on github).
   Navigate to the *CI Pipeline web interface* (hosted on *Github Actions*) and inspect the **build** results!

Note: You might have already *pushed*, in case you answered *yes*, in the *initialize_git_repo* prompt, while generating the Python Package, and in that case, the **Test Workflow** should have already started running! Out-of-the-box, *triggering* the **Test Workflow** happens only when pushing to the *master* or *dev* branch.

References

Here, you will find the `cookiecutter_python`'s **Python API Reference Documentation**.

cookiecutter_python package

Subpackages

cookiecutter_python.backend package

Subpackages

cookiecutter_python.backend.error_handling package

Submodules

cookiecutter_python.backend.error_handling.handler_builder module

class cookiecutter_python.backend.error_handling.handler_builder.**CriticalHandlerBuilder**

Bases: `object`

class cookiecutter_python.backend.error_handling.handler_builder.**HandlerBuilder**

Bases: `object`

subclasses: Dict[str, type] = {'critical': <class 'cookiecutter_python.backend.error_handling.handler_builder.CriticalHandlerBuilder'>, 'non-critical': <class 'cookiecutter_python.backend.error_handling.handler_builder.NonCriticalHandlerBuilder'>}

class cookiecutter_python.backend.error_handling.handler_builder.**NonCriticalHandlerBuilder**

Bases: `object`

Module contents

class cookiecutter_python.backend.error_handling.**HandlerBuilder**

Bases: object

subclasses: Dict[str, type] = {'critical': <class 'cookiecutter_python.backend.error_handling.handler_builder.CriticalHandlerBuilder'>, 'non-critical': <class 'cookiecutter_python.backend.error_handling.handler_builder.NonCriticalHandlerBuilder'>}

cookiecutter_python.backend.generator package

Submodules

cookiecutter_python.backend.generator.generator module

Module contents

cookiecutter_python.backend.hosting_services package

Submodules

cookiecutter_python.backend.hosting_services.check_engine module

class cookiecutter_python.backend.hosting_services.check_engine.**Engine**(*config_file: None | str*, *default_config: None | bool*, *services_info: Tuple*, *checker:* Checker *= NOTHING*, *handlers:* Handlers *= NOTHING*)

Bases: object

check(*servers: List[str]*)

Request Future per supported server, for web hosting service checks
For each server the dedicated 'checker' is called, which tries to return a Future.
Returns None checker's 'activation' boolean flag was off at runtime. Returns None if internal mechanism for determining server URL fails to derive the URL (atm URL is only trie d to be read from User Config yaml)

Parameters
servers (*List[str]*) – [description]
Returns
[description]
Return type
[type]

checker: *Checker*

config_file: None | str

static create(*config_file: str*, *default_config: bool*)

Initialize objects, for Asynchronous http with 3rd-party Services
Objects are designed to Ask PyPI and Read The Docs, if the soon to be generated package name, and readthedocs project slug are available.
These 'Checker' objects make asynchronous http requests to PyPI and RTD web servers, for non-blocking IO, and to avoid blocking the main thread.
Checkers are initialized as 'Activated' if User Config is given and Default Config is False.
Then each Checker (pypi, rtd) requires: - PyPI requires the 'pkg_name' in User's yaml Config - RTD requires the 'readthedocs_project_slug' in User's yaml Config
to derive the URLs for Future Requests

Parameters
- **config_file** (*str*) – user's yaml config file
- **default_config** (*bool*) – default config flag

default_config: None | bool

handle(*request_result*)

handlers: *Handlers*

services_info: Tuple

cookiecutter_python.backend.hosting_services.check_service module

class cookiecutter_python.backend.hosting_services.check_service.**ServiceChecker**(*name_extractor: Callable[[str], str]*, *web_service_checker: WebHostingServiceChecker*, *activate_flag: bool*, *config_file_path: str*)

Bases: object

activate_flag: bool

config_file_path: str

static create(*hosting_service_info*, *activate_flag: bool*, *config_file_path*)

name_extractor: Callable[[str], str]

property service_name

web_service_checker: *WebHostingServiceChecker*

cookiecutter_python.backend.hosting_services.check_web_hosting_service module

class cookiecutter_python.backend.hosting_services.check_web_hosting_service.**WebHostingServiceChecker**(*u*
C
st

Bases: object

Check if CI config is out-of-the-box ok to integrate with hosting services

static create(*hosting_service*)

url_getter: Callable[[str], str]

cookiecutter_python.backend.hosting_services.checker module

class cookiecutter_python.backend.hosting_services.checker.**Checker**(*config_file: None | str*, *default_config: None | bool*, *checkers: Checkers*)

Bases: object

checkers: *Checkers*

config_file: None | str

default_config: None | bool

static from_hosting_info(*config_file*, *default_config*, *hosting_infos*)

Activate Web Host Checks if user config and NOT default config

cookiecutter_python.backend.hosting_services.checkers module

class cookiecutter_python.backend.hosting_services.checkers.**Checkers**(*checkers: MutableMapping*)

Bases: object

static from_hosting_info(*hosting_infos*, *activate_flag*, *config_file*)

cookiecutter_python.backend.hosting_services.exceptions module

exception
cookiecutter_python.backend.hosting_services.exceptions.**ContextVariableDoesNotExist**

Bases: Exception

Raised when a Context Variable does not exist, in a dict-like object.

cookiecutter_python.backend.hosting_services.extract_name module

class cookiecutter_python.backend.hosting_services.extract_name.**NameExtractor**(*name_extractor: Callable[[Mapping], str]*)

Bases: object

Extract Context Value, from a User's Config (YAML) file.

static create(*hosting_service_info*)

name_extractor: Callable[[Mapping], str]

cookiecutter_python.backend.hosting_services.handle_hosting_service_check module

class cookiecutter_python.backend.hosting_services.handle_hosting_service_check.**CheckHostingServiceHandl**

Bases: object

check_hosting_service: Callable[[str], bool]

package_name: str

service_name: str

cookiecutter_python.backend.hosting_services.handler module

class cookiecutter_python.backend.hosting_services.handler.**CheckHostingServiceResultHandler**(*service_name: str*)

Bases: object

static is_future_response_200(*result*) → bool

service_name: str

class cookiecutter_python.backend.hosting_services.handler.**Handlers**(*handlers: Mapping*)

Bases: object

static from_checkers(*checkers*)

handlers: Mapping

cookiecutter_python.backend.hosting_services.value_extractor module

class cookiecutter_python.backend.hosting_services.value_extractor.**BaseValueExtractor**(*key_name: str*)

Bases: *ValueExtractor*

key_name: str

class cookiecutter_python.backend.hosting_services.value_extractor.**ValueExtractor**

Bases: object

cookiecutter_python.backend.hosting_services.web_hosting_service module

class cookiecutter_python.backend.hosting_services.web_hosting_service.**HostingServiceInfo**

Bases: object

create(**args*, ***kwargs*)

Factory method for creating WebHostingService instances.

Raises

NotImplementedError – [description]

property service

property variable_name: str

class cookiecutter_python.backend.hosting_services.web_hosting_service.**HostingServices**

Bases: object

subclasses: Dict[str, type] = {'pypi': <class 'cookiecutter_python.backend.hosting_services.web_hosting_service.PyPIServerFactory'>, 'readthedocs': <class 'cookiecutter_python.backend.hosting_services.web_hosting_service.ReadTheDocsServerFactory'>}

class cookiecutter_python.backend.hosting_services.web_hosting_service.**HostingServicesInfo**(**args*)

Bases: SubclassRegistry[*HostingServiceInfo*]

class cookiecutter_python.backend.hosting_services.web_hosting_service.**PyPIServerFactory**

Bases: *HostingServiceInfo*

create(**args*, ***kwargs*)

Factory method for creating WebHostingService instances.

Raises

NotImplementedError – [description]

property variable_name: str

class cookiecutter_python.backend.hosting_services.web_hosting_service.**ReadTheDocsServerFactory**

Bases: *HostingServiceInfo*

create(**args*, ***kwargs*)

Factory method for creating WebHostingService instances.

Raises

NotImplementedError – [description]

property variable_name: str

class cookiecutter_python.backend.hosting_services.web_hosting_service.**URLGetter**(*url_pattern: str*, *service_name: str*)

Bases: object

service_name: str

url_pattern: str

class cookiecutter_python.backend.hosting_services.web_hosting_service.**WebHostingService**(*url: URLGetter*)

Bases: object

static create(*url_pattern: str*, *service_name: str*)

url: *URLGetter*

Module contents

class cookiecutter_python.backend.hosting_services.**Engine**(*config_file: None | str*, *default_config: None | bool*, *services_info: Tuple*, *checker: Checker = NOTHING*, *handlers: Handlers = NOTHING*)

Bases: object

check(*servers: List[str]*)

Request Future per supported server, for web hosting service checks
For each server the dedicated 'checker' is called, which tries to return a Future.
Returns None checker's 'activation' boolean flag was off at runtime. Returns None if internal mechanism for determining server URL fails to derive the URL (atm URL is only trie d to be read from User Config yaml)

Parameters

servers (*List[str]*) – [description]

Returns

[description]

Return type

[type]

checker: *Checker*

config_file: None | str

static create(*config_file: str*, *default_config: bool*)

Initialize objects, for Asynchronous http with 3rd-party Services
Objects are designed to Ask PyPI and Read The Docs, if the soon to be generated package name, and readthedocs project slug are available.
These 'Checker' objects make asynchronous http requests to PyPI and RTD web servers, for non-blocking IO, and to avoid blocking the main thread.
Checkers are initialized as 'Activated' if User Config is given and Default Config is False.
Then each Checker (pypi, rtd) requires: - PyPI requires the 'pkg_name' in User's yaml Config - RTD requires the 'readthedocs_project_slug' in User's yaml Config
to derive the URLs for Future Requests

Parameters
- **config_file** (*str*) – user's yaml config file
- **default_config** (*bool*) – default config flag

default_config: None | bool

handle(*request_result*)

handlers: *Handlers*

services_info: Tuple

cookiecutter_python.backend.sanitization package

Subpackages

cookiecutter_python.backend.sanitization.string_sanitizers package

Submodules

cookiecutter_python.backend.sanitization.string_sanitizers.base_sanitizer module

class cookiecutter_python.backend.sanitization.string_sanitizers.base_sanitizer.**AbstractSanitizer**

Bases: *SanitizerInterface*, ABC

exception_msg: str

abstract log_message(*error*, *data*) → Tuple

verify: Callable[[Any], None]

class cookiecutter_python.backend.sanitization.string_sanitizers.base_sanitizer.**BaseSanitizer**(*verify: Callable[[A None]*, *exception_msg: str*, *log_func: Callable[[st Any]*, *Tuple]*)

Bases: *AbstractSanitizer*

exception_msg: str

log_message(*error: str*, *data*) → Tuple

verify: Callable[[Any], None]

class cookiecutter_python.backend.sanitization.string_sanitizers.base_sanitizer.**SanitizerInterface**

Bases: ABC

Sanitizer for the Generator Input Parameters.

cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_input module

class cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_input.**BaseSanitizer**(*verify: Callab None], ex- cep- tion_m str, log_fu Callab Any], Tu- ple]*)

Bases: *AbstractSanitizer*

exception_msg: str

log_message(*error: str*, *data*) → Tuple

verify: Callable[[Any], None]

exception cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_input.**InputValueError**

Bases: Exception

cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_module_name module

class cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_module_name.**ModuleNameSanitizer**

Bases: RegExSanitizer

exception_msg = 'Expected a valid Python Module name value'

classmethod log_message(*error*, *module*) → Tuple[str | Mapping, ...]

regex: t.ClassVar[Pattern] = re.compile('^[_a-zA-Z][_a-zA-Z0-9]+$')

cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_module_name.**sanitize_module_nam**

cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_version module

class cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_version.**VersionSanitizer**

Bases: RegExSanitizer

exception_msg = 'Expected a Semantic Version value'

classmethod log_message(*error*, *string*) → Tuple[str | Mapping, ...]

regex: t.ClassVar[Pattern] = re.compile('^(?P<major>0|[1-9]\\d*)\\.(?P<minor>0|[1-9]\\d*)\\.(?P<patch>0|[1-9]\\d*)(?:-(?P<prerelease>(?:0|[1-9]\\d*|\\d*[a-zA-Z-][0-9a-zA-Z-]*)(?:\\.(?:0|[1-9]\\d*|\\d*[a-zA-Z-][0-9a-zA-Z-]*))*))?(?:\\+(?P<bu)

cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_version.**sanitize_version**(*version: str*) → None

Module contents

Submodules

cookiecutter_python.backend.sanitization.input_sanitization module

class cookiecutter_python.backend.sanitization.input_sanitization.**Sanitize**

Bases: object

property exceptions: Mapping[str, Type[Exception] | Tuple]

exceptions_map: Dict[str, List[Type[Exception]]] = {'interpreters': [<class 'cookiecutter_python.backend.sanitization.interpreters_support.InvalidInterpretersError'>], 'module-name': [<class 'cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_input.InputValueError'>], 'semantic-version': [<class 'cookiecutter_python.backend.sanitization.string_sanitizers.sanitize_reg_input.InputValueError'>]}

classmethod register_exception(*sanitizer_identifier: str*)

Add an Exception to the sanitizers' expected exceptions registry.

Parameters

sanitizer_identifier (*str*) – [description]

classmethod register_sanitizer(*sanitizer_identifier: str*)

Add a callback to the sanitizers' registry.

Parameters

sanitizer_identifier (*str*) – [description]

sanitizers_map: Dict[str, Callable[[Any], None]] = {'interpreters': <function verify_input_interpreters>, 'module-name': <function sanitize_module_name>, 'semantic-version': <function sanitize_version>}

cookiecutter_python.backend.sanitization.interpreters_support module

exception
cookiecutter_python.backend.sanitization.interpreters_support.**InvalidInterpretersError**

Bases: Exception

cookiecutter_python.backend.sanitization.interpreters_support.**unsupported_interpreters**(*interpreters: Sequence[str]*) → Iterator[str]

cookiecutter_python.backend.sanitization.interpreters_support.**verify_input_interpreters**(*interpreters: Sequence[str]*) → None

Module contents

Submodules

cookiecutter_python.backend.check_server_result module

class cookiecutter_python.backend.check_server_result.**CheckWebServerResult**

Bases: ABC

Interface for checking the result of a web server request.

abstract property future

abstract property name: str

The name of the resource requested to search on the web server.

Returns

the name of the resource (ie python package slug, rtd project)

Return type

str

abstract property service_name: str

The name of the web server.

Returns

the name (slug) of the web server

Return type

str

cookiecutter_python.backend.gen_docs_common module

Internal configuration for Documentation Generation.

cookiecutter_python.backend.gen_docs_common.**get_docs_gen_internal_config**() → Dict[str, str]

Derive the internal configuration for Documentation Generation.

Information included:

- the folder where we each docs builder will generate the docs. We locate the template folder for each docs builder, which is the Single Source of Truth for the docs builder's output folder.

cookiecutter_python.backend.helpers module

cookiecutter_python.backend.helpers.**parse_context**(*config_file: str*)

cookiecutter_python.backend.load_config module

exception cookiecutter_python.backend.load_config.**InvalidYamlFormatError**

Bases: Exception

exception cookiecutter_python.backend.load_config.**UserYamlDesignError**

Bases: Exception

cookiecutter_python.backend.load_config.**get_interpreters_from_yaml**(*config_file: str*) → Mapping[str, Sequence[str]] | None

Parse the 'interpreters' variable out of the user's config yaml file.

Parameters

config_file (*str*) – path to the user's config yaml file

Raises

- *InvalidYamlFormatError* – if yaml parser fails to load the user's config
- *UserYamlDesignError* – if yaml does not contain the 'default_context' key

Returns

dictionary with intepreters as a sequence of strings,
mapped to the 'supported-interpreters' key

Return type
GivenInterpreters

cookiecutter_python.backend.load_config.**load_yaml**(*config_file*) → MutableMapping

cookiecutter_python.backend.main module

cookiecutter_python.backend.main.**generate**(*no_input=False*, *offline=False*, *extra_context=None*, *replay=False*, *overwrite=False*, *output_dir='.'*, *config_file=None*, *skip_if_file_exists=False*, *default_config=False*, *password=None*, *directory=None*, *checkout=None*) → str

Create Python Project, with CI/CD pipeline, from the project template.
Generate/Scaffold a new Python Project, including configuration enabling automations such as CI and Continuous Delivery of Docker and Python 'artifacts', and Continuous Documentation of the Python Project.

cookiecutter_python.backend.post_main module

exception cookiecutter_python.backend.post_main.**CheckWebServerError**

Bases: Exception
Raised on Connection Error, when Requesting a Web Server's Future.

cookiecutter_python.backend.post_main.**post_main**(*request*)

Check if any CI 'deployment' (ie in pypi), would require minor tweak.

cookiecutter_python.backend.pre_main module

cookiecutter_python.backend.pre_main.**pre_main**(*request*)

Do preparatory steps Generation process, by settings things as the Template Context.

Parameters
****kwargs** – Arbitrary keyword arguments.

cookiecutter_python.backend.proxy module

class cookiecutter_python.backend.proxy.**BaseProxy**(*proxy_subject: ProxySubject*)

Bases: Proxy[T], Generic[T]

static log_info_args(*message: str*, **args*, ***kwargs*) → Tuple[str, str]

cookiecutter_python.backend.request module

class cookiecutter_python.backend.request.**Request**(***, *config_file: str*, *default_config: bool*, *web_servers: List[str]*, *no_input: bool*, *extra_context: dict*, *check: Any | None = None*, *check_results: None | Iterable[CheckWebServerResult] = None*, *offline: bool = False*)

Bases: object

check: Any

check_results: None | Iterable[*CheckWebServerResult*]

config_file: str

default_config: bool

extra_context: dict
no_input: bool
offline: bool
web_servers: List[str]

cookiecutter_python.backend.user_config_proxy module

Module contents

exception cookiecutter_python.backend.**CheckWebServerError**

Bases: Exception

Raised on Connection Error, when Requesting a Web Server's Future.

cookiecutter_python.backend.**generate**(*no_input=False*, *offline=False*, *extra_context=None*, *replay=False*, *overwrite=False*, *output_dir='.'*, *config_file=None*, *skip_if_file_exists=False*, *default_config=False*, *password=None*, *directory=None*, *checkout=None*) → str

Create Python Project, with CI/CD pipeline, from the project template.

Generate/Scaffold a new Python Project, including configuration enabling automations such as CI and Continuous Delivery of Docker and Python 'artifacts', and Continuous Documentation of the Python Project.

cookiecutter_python.backend.**get_docs_gen_internal_config**() → Dict[str, str]

Derive the internal configuration for Documentation Generation.

Information included:

- the folder where we each docs builder will generate the docs. We locate the template folder for each docs builder, which is the Single Source of Truth for the docs builder's output folder.

cookiecutter_python.handle package

Subpackages

cookiecutter_python.handle.dialogs package

Subpackages

cookiecutter_python.handle.dialogs.lib package

Submodules

cookiecutter_python.handle.dialogs.lib.project_name module

class cookiecutter_python.handle.dialogs.lib.project_name.**ProjectNameDialog**

Bases: object

dialog(*cookie_vars*) → Mapping[str, str]

Module contents

class cookiecutter_python.handle.dialogs.lib.**InteractiveDialog**

Bases: object

subclasses = {'project-name': <class 'cookiecutter_python.handle.dialogs.lib.project_name.ProjectNameDialog'>}

Submodules

cookiecutter_python.handle.dialogs.dialog module

class cookiecutter_python.handle.dialogs.dialog.**Dialog**

Bases: object

abstract dialog(**args*, ***kwargs*)

class cookiecutter_python.handle.dialogs.dialog.**DialogRegistry**(**args*)

Bases: SubclassRegistry[*Dialog*]

class cookiecutter_python.handle.dialogs.dialog.**InteractiveDialog**

Bases: object

subclasses = {'project-name': <class 'cookiecutter_python.handle.dialogs.lib.project_name.ProjectNameDialog'>}

Module contents

class cookiecutter_python.handle.dialogs.**InteractiveDialog**

Bases: object

subclasses = {'project-name': <class 'cookiecutter_python.handle.dialogs.lib.project_name.ProjectNameDialog'>}

Submodules

cookiecutter_python.handle.interactive_cli_pipeline module

Handles sequence of Interactive User Dialogs, for Context Information.

class cookiecutter_python.handle.interactive_cli_pipeline.**InteractiveDialogsPipeline**

Bases: object

Handles sequence of Interactive User Dialogs, for Context Information.

dialogs = ['project-name']

process(*request*)

Process sequence of Interactive User Dialogs, for Context Information.

cookiecutter_python.handle.node_base module

class cookiecutter_python.handle.node_base.**DialogNode**(*dialog*)

Bases: *Node*[List, Mapping[str, Any]]

Handles a single Interactive User Dialog, for Context Information.

process(*request*)

Process a single Interactive User Dialog, for Context Information.

cookiecutter_python.handle.node_factory module

class cookiecutter_python.handle.node_factory.**NodeFactory**

Bases: object

static create(*dialog_name: str*)

cookiecutter_python.handle.node_interface module

class cookiecutter_python.handle.node_interface.**Node**

Bases: ABC, Generic[T, TT]

abstract process(*request: T*) → TT | None

Module contents

cookiecutter_python.hooks package

Submodules

cookiecutter_python.hooks.post_gen_project module

Post Cookie Hook: Templated File with jinja2 syntax
Cookiecutter post generation hook script that handles operations after the template project is used to generate a target project.

cookiecutter_python.hooks.post_gen_project.**CLI_ONLY**(*x*)

cookiecutter_python.hooks.post_gen_project.**PYTEST_PLUGIN_ONLY**(*x*)

exception cookiecutter_python.hooks.post_gen_project.**PostFileRemovalError**

Bases: Exception

cookiecutter_python.hooks.post_gen_project.**get_context**() → OrderedDict

Get the Context, that was used by the Templating Engine at render time

cookiecutter_python.hooks.post_gen_project.**get_request**()

cookiecutter_python.hooks.post_gen_project.**git_commit**(*request*)

Commit the staged changes in the generated project.

cookiecutter_python.hooks.post_gen_project.**initialize_git_repo**(*project_dir: str*)

Initialize the Git repository in the generated project.

cookiecutter_python.hooks.post_gen_project.**is_git_repo_clean**(*project_directory: str*) → bool

Check to confirm if the Git repository is clean and has no uncommitted changes. If its clean return True otherwise False.

cookiecutter_python.hooks.post_gen_project.**iter_files**(*request*)

cookiecutter_python.hooks.post_gen_project.**main**()

Delete irrelevant to Project Type files and optionally do git commit.

cookiecutter_python.hooks.post_gen_project.**post_file_removal**(*request*)

Preserve only files relevant to Project Type requested to Generate.
Delete files that are not relevant to the project type requested to generate.
For example, if the user requested a 'module' project type, then delete the files that are only relevant to a 'module+cli' project.

Parameters

request (*[type]*) – [description]

cookiecutter_python.hooks.post_gen_project.**post_hook**()

Delete irrelevant to Project Type files and optionally do git commit.

cookiecutter_python.hooks.post_gen_project.**run_process_python36_n_below**(**args*, ***kwargs*)

cookiecutter_python.hooks.post_gen_project.**run_process_python37_n_above**(**args*, ***kwargs*)

cookiecutter_python.hooks.post_gen_project.**subprocess_run**(**args*, ***kwargs*)

cookiecutter_python.hooks.pre_gen_project module

Pre Cookie Hook: Templated File with jinja2 syntax

exception cookiecutter_python.hooks.pre_gen_project.**InputSanitizationError**

Bases: Exception

cookiecutter_python.hooks.pre_gen_project.**get_request**()

cookiecutter_python.hooks.pre_gen_project.**hook_main**(*request*)

cookiecutter_python.hooks.pre_gen_project.**input_sanitization**(*request*)

cookiecutter_python.hooks.pre_gen_project.**main**()

Module contents

Submodules

cookiecutter_python.cli module

Main *cookiecutter_python* CLI.

cookiecutter_python.cli.**version_msg**()

Message about Python Generator version, location and Python version.

cookiecutter_python.cli_handlers module

cookiecutter_python.cli_handlers.**handle_error**(*error*)

cookiecutter_python.exceptions module

cookiecutter_python.exceptions.**error_2_str**(*error*)

cookiecutter_python.utils module

cookiecutter_python.utils.**load**(*interface: Type[T]*, *module: str | None = None*) → List[Type[T]]

Dynamically import all class objects that implement the given interface.
The classes (class objects) are discovered and imported in the namespace, by searching within each module found inside the input 'dire' (path) directory.
Each class object is an attribute found in a module's namespace. We classify an attribute as a (correct) "class to import", if the following python boolean expression evaluates to True:
isclass(attribute) and issubclass(attribute, interface)
If 'dire' is not given then we consider the modules that are inside the same directory as the one where the module of the invoking code resides.

Parameters

- **interface** (*Type[T]*) – the type (ie class) that the imported classes should 'inherit' (subclass) from
- **module** (*str*) – module containing the modules to inspect. Defaults to the same module (directory) as the one where the module of the invoking code resides.

Module contents

Developer's Corner

Here we present the **Software Architecture** and offer **Guides** on how to leverage the **CI/CD** to do various Development Operations, in a **GitOps** way.

CI/CD Pipeline

CI/CD Pipeline

Architecture

Software Architecture

Here you can find the software architecture of the project.

Module Dependencies

Here you can find the dependencies between the modules of the project.
The dependencies are Visualized as a Graph, where Nodes are the modules and the Edges are python `import` statements.
The dependencies are visualized, after running the following command:

```
tox -e pydeps
```

First-party Dependencies

All Dependencies - A

All Dependencies - B

All Dependencies - C

GitOps Guides

How to use our `System` for **Automated Git Ops.**

Streamline Documentation Updates

1. Branch of off *main* Branch, and checkout your *topical branch* (*tb*).
2. Create Docs-only changes and commit them to your *tb*.
3. Push git tag *quick-release*, to trigger the Docs Release Workflow, on the CI

A new PR, is expected to **open** from *tb* to a *dedicated docs* branch, and automatically **merge** if Docs Build passed on *rtd* CI.
Then, a new PR, is expected to **open** from *dedicated docs* branch to *main*, with extra commits with Sem Ver Bump, and Changelog updates.

4. Wait for second PR to open, go to github web IU to review it, and merge it.

A new **tag** is expected to be created (on the new main/master commit), and a *PyPI* distribution will be uploaded, a new Docker Image on Dockerhub, and a new Github Release will be created.

Workflows References

- quick-docs.yaml : Listens to `quick-release` git tag, and merges tb –> db, after opening PR.

Release Candidate / Test Deployment

From your branch, run

```
rc_tag=$(grep -E -o '^version\s*=\s*\".*\"' pyproject.toml | cut -d'"' -f2)
rc_tag="${rc_tag}-rc"

git tag "$git_tag" || (git tag -d "$git_tag" && git tag "$git_tag")
git push origin -d "$git_tag"; git push origin "$git_tag"
```

This will, trigger the CI/CD Pipeline and instruct it to do a `Test Deployment`.

Test Deployment is a full deployment of the package to the test environment.
And is the closest thing to a real (production) deployment.

The CI/CD Pipeline will:

1. make wheel builds (and unit test them) for the package using a Job Matrix factoring OS x Py Versions
2. Do as normal measuring of Code Coverage, Static Code Analysis, Docker Build
3. Publish Python Wheel Distribution in Test Environment, at test.pypi.org

Docker Build

Dockerfile - Build Process

1.5 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

D

E

F

N

O

P

R

S

U

V

W