Computational Molecular Sciences Cookiecutter Documentation

Levi N. Naden, Daniel G. A. Smith

Nov 03, 2020

Contents

Note: These docs are are mainly a recreation of the package's `README.md` file as an example of building the docs. Feel free to borrow from this example and consider splitting the docs into multiple pages!

A cookiecutter template for those interested in developing computational molecular sciences packages in Python. Skeletal starting repositories can be created from this template to create the file structure semi-autonomously so you can focus on what's important: the science!

The skeletal structure is designed to help you get started, but do not feel limited by the skeleton's features included here. Just to name a few things you can alter to suite your needs: change continuous integration options, remove deployment platforms, or test with a different suite.

CHAPTER 1

Features

- Python-centric skeletal structure with initial module files
- Pre-configured `setup.py` for installation and packaging
- Pre-configured Window, Linux, and OSX continuous integration on AppVeyor and Travis-CI
- Choice of dependency locations through `conda-forge`, default `conda`, or `pip`
- Basic testing structure with PyTest
- Automatic `git` initialization + tag
- GitHub Hooks
- Automatic package version control with Versioneer
- Sample data inclusion with packaging instructions
- Basic documentation structure powered by Sphinx
- Automatic license file inclusion from several common Open Source licenses (optional)

CHAPTER 2

Requirements

- Python 3.6, or 3.7
- Cookiecutter
- Git

CHAPTER 3

Usage

With cookiecutter installed, execute the following command inside the folder you want to create the skeletal repository.

```
cookiecutter gh:molssi/cookiecutter-cms
```

Which fetches this repository from github automatically and prompts the user for some simple information such as package name, author(s), and licences.

CHAPTER 4

Next steps and web integrations

The repository contains a number of "hooks" that integrate with a variety of web services. To fully integrate the project with these web services and to get started developing your project please proceed through the following directions.

4.1 Local installation

For development work it is often recommended to do a "local" python install via `pip install -e .`. This command will insert your new project into your Python site-packages folder so that it can be found in any directory on your computer.

4.2 Setting up with GitHub

Upon creation, this project will initialize the output as a `git` repository compatible with Versioneer. However, this does not automatically register the repository with GitHub. To do this, follow the instructions for Adding an existing project to GitHub using the command line. Follow the first step to create the repository on GitHub, but ignore the warnings about the README, license, and `.gitignore` files as this repo creates them. From there, you can skip to after the "first commit" instructions and proceed from there.

4.3 Testing

The Python testing framework was chosen to be pytest for this project. Other testing frameworks are available; however, the authors believe the combination of easy parametrization of tests, fixtures, and test marking make `pytest` particularly well suited for molecular software packages.

To get started additional tests can be added to the `project/tests/` folder. Any function starting with `test_*` will automatically be included in the testing framework. While these can be added in anywhere in your directory structure, it is highly recommended to keep them contained within the `project/tests/` folder.

Tests can be run with the `pytest -v` command. There are a number of additional command line arguments to explore.

4.4 Continuous Integration

Testing is accomplished with both Appveyor (for Windows testing) and Travis-CI (for Linux testing). These frameworks are chosen as they are completely free for open source projects and allow you to automatically verify that your project works under a variety of OS's and Python versions. To begin please register with both Appveyor and Travis-CI and turn on the git hooks under the project tabs. You will also want to correct the badges which appear on the output README file to point to the correct links

Documentation Make a ReadTheDocs account and turn on the git hook. Although you can manually make the documentation yourself through Sphinx, you can also configure ReadTheDocs to automatically build and publish the documentation for you. The initial skeleton of the documentation can be found in the `docs` folder of your output.

4.5 Additional Python Settings in `setup.py`

This Cookiecutter generates the package, but there are a several package-specific Python settings you can tune to your package's installation needs. These are settings in the `setup.py` file which contains instructions for Python on how to install your package. Each of the options in the file are commented with what it does and when it should be used.

CHAPTER 5

Why is Python 2.X not on the supported versions?

New projects generally should not be built with Python 2.7 support in mind, see the Python 3 Statement. Although the final Python 2.7 release will be supported through 2020 and is the default on many legacy systems, Python 3 has been released for almost a decade and projects long term usage should not be shacked by legacy methods that will have to be replaced in very short order as Python 2 support is retired.

CHAPTER 6

Output Skeleton

This will be the skeleton made by this `cookiecutter`, the items marked in `{{ }}` will be replaced by your choices upon setup.

```
.
├── LICENSE                          <- License file
├── README.md                        <- Description of project which GitHub will render
├── appveyor.yml                     <- AppVeyor config file for Windows testing (if␣
↪chosen)
├── {{repo_name}}
│   ├── __init__.py                  <- Basic Python Package import file
│   ├── {{first_module_name}}.py     <- Starting packge module
│   ├── data                         <- Sample additional data (non-code) which can be␣
↪packaged
│   │   ├── README.md
│   │   └── look_and_say.dat
│   ├── tests                        <- Unit test directory with sample tests
│   │   ├── __init__.py
│   │   └── test\_{{repo_name}}.py
│   └── _version.py                  <- Automatic version control with Versioneer
├── devtools                         <- Deployment, packaging, and CI helpers directory
│   ├── README.md
│   ├── conda-recipe                 <- Conda build and deployment skeleton
│   │   ├── bld.bat
│   │   ├── build.sh
│   │   └── meta.yaml
│   └── travis-ci
│       └── install.sh
├── docs                             <- Documentation template folder with many␣
↪settings already filled in
│   ├── Makefile
│   ├── README.md                    <- Instructions on how to build the docs
│   ├── _static
│   ├── _templates
│   ├── conf.py
```

(continues on next page)

(continued from previous page)

```
│   ├── index.rst
│   └── make.bat
├── setup.cfg                       <- Near-master config file to make house INI-like␣
↪settings for Coverage, Flake8, YAPF, etc.
├── setup.py                        <- Your package's setup file for installing with␣
↪additional options that can be set
├── versioneer.py                   <- Automatic version control with Versioneer
├── .github                         <- GitHub hooks for user contrubtion and pull␣
↪request guides
│   ├── CONTRIBUTING.md
│   └── PULL_REQUEST_TEMPLATE.md
├── .codecov.yml                    <- Codecov config to help reduce its verbosity to␣
↪more reasonable levels
├── .gitignore                      <- Stock helper file telling git what file name␣
↪patterns to ignore when adding
└── .travis.yml                     <- Travis-CI config file for Linux and OSX testing
```

CHAPTER 7

Additional Pages

7.1 Warnings and Caveats from the Cookiecutter

We encourage users to look at the parent Computational Molecular Sciences Cookiecutter as ways to template their own output projects from the Cookiecutter. However, there are a few things the parent does to make the illustration work, but should probably not be followed in your projects. These are mostly because the parent has to simulate an output, then test the output of the cookiecutter, which is something you will not have to do with your project... Unless you are making a Cookiecutter which makes Cookiecutters, but that is beyond the scope of this project.

7.1.1 Continuous Integration (CI) Caveats

The parent Cookiecutter must emulate the the process of creating and running tests, while in its own tests. Since Travis and AppVeyor are not intended to do this, we have to do some trickery to manually process the YAML output files after executing the Cookiecutter. This is something you, the user of this Cookiecutter, should not have to worry about and can instead just use Travis and AppVeyor as those programs intend.

7.1.2 Writing helpful documentation

The primary documentation for this Cookiecutter is mostly just a copy of the main README.md file. Your docs should be more detailed in ways the README.md cannot. The README.md file is rendered by GitHub, but will (should) not contain all of the detailed instructions, settings, applications, benchmarking which can be elaborated on in full documentations.

CHAPTER 8

Acknowledgments

This cookiecutter is developed by Levi N. Naden and Daniel G. A. Smith from the Molecular Sciences Software Institute (MolSSI). Copyright (c) 2018.

Directory structure template based on recommendation from the Chodera Lab's Software Development Guidelines.

Original hosting of repository owned by the Chodera Lab

Elements of this repository drawn from the cookiecutter-data-science by Driven Data and the MolSSI Python Template.