cookiecutter-chemios Documentation

Release 0.9.0

Kobi Felton

Jun 23, 2018

Contents

1 User's Guide **3**
- 1.1 Introduction . . . 3
- 1.2 Installation (5 minutes) . . . 4
- 1.3 Getting Started . . . 5
- 1.4 Next Steps . . . 6

2 Additional Notes **9**
- 2.1 Prompts . . . 9
- 2.2 Travis/PyPI Setup . . . 10

Cookiecutter-Chemios is a template for chemios laboratory instrument drivers.

Chemios is a platform automating and monitoring laboratory instruments. So far, we have written drivers for syringe pumps, temperature controllers and spectrometers. Cookiecutter Chemios standardizes the format for new instrument drivers.

CHAPTER 1

User's Guide

1.1 Introduction

1.1.1 Benefits

- **faster developement**: All the files and boilerplate for a new instrument driver can be created with one command.
- **it's free**: This code is free and open source, so you can easily create and share drivers you write (unlike LabView)
- **standardization**: Each instrument driver has a standard interface for all brands, making it easy to switch between brands without rewriting code (e.g, switch betwen Harvard Apparatus and New Era syringe pumps).
- **cross platform**: The code is written in Python which is compatible with devices running Mac OS, Windows or Linux (including the Raspberry Pi).
- **command line interface**: Each driver includes a command line interface for quickly testing a connection to an instrument or logging data to a file.
- **maintanibility**: Best practices for driver development such as unit testing, continuous integration and autodocumentation are included.
- **packaging**: Every driver can be published to PyPi and installed with a single `pip install` command.

1.1.2 Structure

Each instance of a chemios instrument driver has a standard structure. Here are some of the most important files in the temperature controllers driver as an example:

```
chemios_tc/ #the folder for the chemios_tc package
    __init__.py
    chemios_tc_base.py #Contains template class for temperature controllers
    Omega9300Series.py #One type of a temperature controller
    cli.py #Source code for the commnad line interface
    utils.py #Useful Utilities
```

(continues on next page)

(continued from previous page)

```
docs/ #Contains file for building documentation
tests/ #Unit tests of the driver
new_instrument.py #Create the boilerplate for a new temperature controller
requirements_dev.txt #Dependencies needed for development
setup.py #Instructions for pacakaging the chemios_tc package
```

Some key benefits of this structure:

- Once you run `python setup.py`, your instrument driver can be imported into a python file. For example with temperature controllers:`from chemios_tc import Omega9300Series`
- Running `new_instrument.py INSTRUMENT_NAME` will create a new file inside the package based on the template in base.py. It will also add the new class in that file to the package level (via __init__.py).

1.2 Installation (5 minutes)

Estimated Time: 5 minutes (if you already have python installed).

1.2.1 Step 1: Install Prerequesites

You will need python 3.0 or above for this tutorial. We suggest installing python via conda, so you can use the conda package manager.

First, you need to create and activate a virtual environment, which will ensure a consistent development environment. Here is how to do it with conda, but you can also use virtualenv.

```
conda create -n YOUR_ENVIRONMENT_NAME
source activate YOUR_ENVIRONMENT_NAME
```

Then, install cookiecutter, which is a python library for creating projects from templates.

```
pip install -U cookiecutter
```

1.2.2 Step 2: Create a driver repository

To create a new chemios driver repository, run the following command. You will be prompted at the command line for the information needed to create the new driver repository. See this *Prompts* for a description of the prompts.

```
cookiecutter https://github.com/Chemios/cookiecutter-chemios
```

1.2.3 Step 3: Install Dev Requirements

Change into the new directory for the instrument driver and run the following command. Make sure your virtual environment is still activated (you should see the name of the environemnt at your command prompt)

```
pip install -r requirements_dev.txt
```

Next, we'll take a look at the struture of your new chemios driver.

1.3 Getting Started

1.3.1 base.py as a template

The heart of each chemios driver is a base.py file. This file contains an abstract base class that models the interface for all instruments of that type.

For example, the temperature controllers base file has a set_temperature and get_current_temperature method: Each temperature controller must implement the set_temperature and get_current_temperature methods.

```
# -*- coding: utf-8 -*-

"""Base Class for Temperature Controllers"""

from abc import ABC

class TemperatureControllers(ABC):
    '''Base Class for Temperature Controllers
    '''
    def get_current_temperature(self):
        ''' Method to get the current temperature
        Returns:
            update: dict

            update = {
                    'temp_set_point': setpoint in °C,
                    'current_temp': temperature in °C
                    }
        '''

    def set_temperature(self, temp_set_point):
        ''' Method to set the temperature
        Args:
            temp_set_point (float): temperature setpoint in °C
        Returns:
            update: dict

            update = {
                    'temp_set_point': setpoint in °C,
                    'current_temp': temperature in °C
                    }
        '''
```

The standard inteface is enforced by each temperature controller inheriting the TemperatureControllers base class. See Omega9300Series.py for an example.

1.3.2 New Instrument

To create a new instrument, we will use the **'new_instrument.py'_** script.

This script creates a new instrument file based on the base file. It also imports the newly created class inside that file to the package level (via __init__.py)

1.4 Next Steps

1.4.1 Create a GitHub Repo

Go to your GitHub account and create a new repo named `mypackage`, where `mypackage` matches the `[project_slug]` from your answers to running cookiecutter. This is so that Travis CI and pyup.io can find it when we get to Step 5.

`If your virtualenv folder is within your project folder, be sure to add the virtualenv folder name to your .gitignore file.`

You will find one folder named after the `[project_slug]`. Move into this folder, and then setup git to use your GitHub repo and upload the code:

```
cd mypackage
git init .
git add .
git commit -m "Initial skeleton."
git remote add origin git@github.com:myusername/mypackage.git
git push -u origin master
```

Where `myusername` and `mypackage` are adjusted for your username and package name.

You'll need a ssh key to push the repo. You can Generate a key or Add an existing one.

1.4.2 Set Up Travis CI

Travis CI org*[0] is a continuous integration tool used to prevent integration problems. Every commit to the master branch will trigger automated builds of the application.

Login using your Github credentials. It may take a few minutes for Travis CI to load up a list of all your GitHub repos. They will be listed with boxes to the left of the repo name, where the boxes have an `X` in them, meaning it is not connected to Travis CI.

Add the public repo to your Travis CI account by clicking the `X` to switch it "on" in the box next to the `mypackage` repo. Do not try to follow the other instructions, that will be taken care of next.

In your terminal, your virtualenv should still be activated. If it isn't, activate it now. Run the Travis CLI tool to do your Travis CI setup:

```
travis encrypt --add deploy.password
```

This will:

- Encrypt your PyPI password in your Travis config.
- Activate automated deployment on PyPI when you push a new tag to master branch.

See *Travis/PyPI Setup* for more information.

1.4.3 Set Up ReadTheDocs

ReadTheDocs hosts documentation for the open source community. Think of it as Continuous Documentation.

Log into your account at ReadTheDocs . If you don't have one, create one and log into it.

[0] For private projects go to Travis CI com

If you are not at your dashboard, choose the pull-down next to your username in the upper right, and select "My Projects". Choose the button to Import the repository and follow the directions.

In your GitHub repo, select Settings > Webhooks & Services, turn on the ReadTheDocs service hook.

Now your documentation will get rebuilt when you make documentation changes to your package.

1.4.4 Set Up pyup.io

pyup.io is a service that helps you to keep your requirements files up to date. It sends you automated pull requests whenever there's a new release for one of your dependencies.

To use it, create a new account at pyup.io or log into your existing account.

Click on the green `Add Repo` button in the top left corner and select the repo you created in Step 3. A popup will ask you whether you want to pin your dependencies. Click on `Pin` to add the repo.

Once your repo is set up correctly, the pyup.io badge will show your current update status.

1.4.5 Release on PyPI

The Python Package Index or PyPI is the official third-party software repository for the Python programming language. Python developers intend it to be a comprehensive catalog of all open source Python packages.

When you are ready, release your package the standard Python way.

See PyPI Help for more information about submitting a package.

Here's a release checklist you can use: https://gist.github.com/audreyr/5990987

CHAPTER 2

Additional Notes

2.1 Prompts

When you create a package, you are prompted to enter these values.

2.1.1 Templated Values

The following appear in various parts of your generated project.

developer_name Your full name.

email Your email address.

github_username Your GitHub username.

instrument_type The name your new instrument type. The plural form is preferred here (e.g., temperature controllers, pumps, etc.).

project_name (filled automatically) The name of your new instrument driver. This is used in documentation, so spaces and any characters are fine here.

project_slug (filled automatically) The namespace of your in. This should be Python import-friendly. Typically, it is the slugified version of project_name.

project_short_description (filled automatically) A 1-sentence description of what your drie does.

pypi_username Your Python Package Index account username.

version The starting version number of the package.

2.1.2 Options

The following package configuration options set up different features for your project.

use_pytest If yes, this will set up a pytest runner for completing test

use_pypi_deployment_with_travis Whether to use PyPI deployment with Travis CI. This allows you to automatically package your driver, so it can be installed via pip.

command_line_interface Whether to create a console script using Click. Console script entry point will match the project_slug. Options: [‘Click’, “No command-line interface”]

open_source_license Choose MIT here.

2.2 Travis/PyPI Setup

Optionally, your package can automatically be released on PyPI whenever you push a new tag to the master branch.

2.2.1 Install the Travis CLI tool

This is OS-specific.

Mac OS X

We recommend the Homebrew travis package:

` brew install travis `

Windows and Linux

Follow the official Travis CLI installationinstructions for your operating system:

https://github.com/travis-ci/travis.rb#installation

2.2.2 How It Works

Once you have the *travis* command - line tool installed, from the root of your project do:

```
travis encrypt --add deploy.password
```

This will encrypt your locally-stored PyPI password and save that to your *.travis.yml* file. Commit that change to git.

2.2.3 Your Release Process

If you are using this feature, this is how you would do a patch release:

```
bumpversion patch
git push --tags
```

This will result in:

- mypackage 0.1.1 showing up in your GitHub tags/releases page
- mypackage 0.1.1 getting released on PyPI

You can also replace patch with *minor* or *major*.

2.2.4 More Details

You can read more about using Travis for PyPI deployment at: https://docs.travis-ci.com/user/deployment/pypi/