oapkg Documentation

Release 0.1.0

revesansparole

Jan 14, 2018

Contents

Contents:

CHAPTER 1

Overview

Default template for openalea packages.

Maybe a bit more description could be usefull.

CHAPTER 2

Installation

2.1 For Users

The easiest method is to directly install the package hosted on PyPI:

```
$ pip install oapkg
```

2.2 For Developers

We recommend using a dedicated conda environment when installing a new package, so create it if you haven't done it already:

```
$ conda create -n myenv python
```

Then activate it:

```
$ activate myenv
```

Download (or clone) the source and then, at the command line:

```
(myenv) $ conda env update --file requirements.yml
(myenv) $ conda env update --file dvlpt_requirements.yml
(myenv) $ python setup.py develop
```

If conda fails to install a package, it usually means that one of the required package cannot be installed with conda. There is three possibilities:

- the package is developed by the openalea community: follow instruction associated to the package that can be found on openalea
- the package is an external package: try using pip to install it instead of conda.
- the package is not available on the default channel. You can try using a less esoteric package next time :)

2.2.1 Run test suite

- Use pytest to run all unit tests associated to the package:

```
(myenv) $ pytest
```

  By default, coverage is activated and will list the lines of codes which are currently not covered by your tests.

- Use pytest with the 'runslow' option to run all tests including functional tests that may require more time to run:

```
(myenv) $ pytest --runslow
```

2.2.2 Compile documentation on your computer

Continuous integration will take care of compiling your documentation automatically to ensure the web version of the documentation is always accurate. However if you want to launch sphinx and compile the documentation on your computer for a quick review purpose for example use the 'build_sphinx' option of 'setup.py':

```
(myenv) $ python setup.py build_sphinx
```

This will create some files in build/sphinx/html. Open the 'index.html' to access the main page of the documentation in a browser.

CHAPTER 3

Usage

Something to say?

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

4.1.4 Write Documentation

itkpkg could always use more documentation, whether as part of the official **itkpkg** docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *itkpkg* for local development.

1. Fork the *itkpkg* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/itkpkg.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenv installed, this is how you set up your fork for local development:

```
$ virtualenv dvlpt
$ dvlpt/script/activate
(dvlpt)$ python setup.py develop
```

4. Create a branch for local development (wip stands for work in progress):

```
(dvlpt)$ git checkout -b wip_name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
 (dvlpt)$ cd itkpkg
 (dvlpt) itkpkg$ flake8
 (dvlpt) itkpkg$ pytest

 (dvlpt) itkpkg$ tox

To get flake8 and tox, just pip install them into your virtualenv.
```

6. Commit your changes and push your branch to GitHub

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin wip_name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 36.

   Check Travis and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ pytest test/test_XXX
```

CHAPTER 5

Credits

5.1 Development Lead

- revesansparole, <revesansparole@gmail.com>

5.2 Contributors

- jchopard <jerome.chopard@itk.fr>

CHAPTER 6

History

CHAPTER 7

Developper's documentation

7.1 src

7.1.1 oapkg package

Submodules

oapkg.config module

oapkg.config.**check**(*cfg*)
Check the validity of parameters in working environment.

Parameters **cfg** (*Config*) – current package configuration

Returns list of faulty parameters

Return type (list of str)

oapkg.config.**require**(*purpose*, *cfg*)
List of requirements for this option for a given purpose.

Parameters

- **purpose** (*str*) – either 'option', 'setup', 'install' or 'dvlpt'
- **cfg** (*Config*) – current package configuration

Returns (list of Dependency)

oapkg.config.**update_parameters**(*cfg*)
Update config with parameters necessary for this option.

Notes: create a section with option name to store params.

Parameters **cfg** (*dict*) – dict of option parameters as seen in pkg_cfg.json

Returns update in place

Return type None

oapkg.handlers module

Used to extend Jinja2 environment with extra arguments

oapkg.handlers.**environment_extensions**(*cfg*)
Add more functionality to an environment.

Parameters **cfg** (*Config*) – current package configuration

Returns any

Return type dict of str

oapkg.handlers.**installed_options**(*cfg*)

oapkg.version module

oapkg.version.**MAJOR = 0**
(int) Version major component.

oapkg.version.**MINOR = 1**
(int) Version minor component.

oapkg.version.**POST = 0**
(int) Version post or bugfix component.

Module contents

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Back to OpenAlea home page.

Python Module Index

o

Index