Documentation

CFEL Controlled Molecule Imaging

2020-05-20

Contents

This package serves as a template for CMI Python projects.

CHAPTER 1

User documentation

1.1 Installing CMItemplate

1.1.1 Prerequisites and obtaining CMItemplate

Since CMItemplate is written in Python, you need to install Python (version 3).

In general, you should specify which Python extension packages are needed, such as

- cmiext
- NumPy

CMItemplate is avaiable on GitHub, please contact Jochen Küpper <jochen.kuepper@cfel.de> for further details.

1.1.2 Installing CMItemplate

A normal installation is performed by simply running the command:

```
python setup.py install
```

However, often you do not have the administrative rights to install in global directories, or simply do not want to overrride a global installtion. In this case, you might want to perform a local installation in your user directory using:

```
python setup.py install --user
```

A similar setup can be achieved using:

```
python setup.py develop --user
```

which, however, sets up the installation in such a way that changes to your source directory are automatically and immediately visible through the installed version. This avoids repeated re-installs while you are developing code.

Once you are satisfied with your changes you might consider reinstalling using one of the above two options.

Fur further details of `develop` install, see http://naoko.github.io/your-project-install-pip-setup

1.1.3 Installing CMItemplate: in user-specified path

Use PYTHONUSERBASE to specify the installation path:

```
setenv PYTHONUSERBASE $HOME/.local
python setup.py install --user
```

In the above example of installation (in tcsh shell), the module will be installed in the following path:

```
$HOME/.local/lib/python/site-packages
```

and the scripts will be installed in the following path:

```
$HOME/.local/bin
```

To import modules and call scripts of such user-specific installation, the following environment declarifications are required:

```
setenv PATH /opt/local/bin:$HOME/.local/bin:$PATH
setenv PYTHONUSERBASE $HOME/.local
```

The above example is provided for the tcsh shell. You can also then use `site` module of python in python command prompt to make sure the environment is properly set up. For example:

```
>>> import site
>>> site.USER_BASE
'$HOME/.local'
```

Also type "which `name of script file`" to find the real path of the script called. It should be in "$HOME/.local/bin".

For further details, see https://docs.python.org/3/install/index.html#inst-alt-install-user and https://docs.python.org/3/using/cmdline.html#envvar-PYTHONUSERBASE

1.2 CMItemplate user guide

The template package is developed and maintained by the Controlled Molecule Imaging group (CMI) at the Center for Free-Electron Laser Science (CFEL), Hamburg, Germany.

Here you should provide further general information, such as scientific publications of the program. See, for instance [Küpper2020].

1.2.1 General usage

The following provides examples of the general usage.

1.2.2 cmitemplate_calc

A (command line) script file called *cmitemplate_calc* is provided as a driver for the calculation. Here you would document its command-line options (ore refer to more extended documentation).

CHAPTER 2

Developer documentation

The full documentation of the source code is available in the following subpages. This includes, for instance, available classes and methods, and their respective parameters and return values.

2.1 cmitemplate package

2.1.1 Submodules

2.1.1.1 cmitemplate.template module

2.1.2 Module contents

2.2 CMItemplate developer guide

Here are a few simple guidelines to please be obeyed when working on CMIdiffract

- Document your code!
  - Use spinx-compatible docstrings to document all classs, methods, functions, etc.
- Write code that is compatible with the latest stable Python 3.x version.
- Make use of NumPy as much as possible.

2.2.1 Source code formatting

- CMI Python projects use the 4-spaces standard for indentation of blocks.
- Do not use tabs, always expand to spaces.
- Try to not extend lines beyond 100 characters
- Keep the utf-8 coding directive in the first line

- Keep the Emacs local variables section at the end of all files, and try to stick to the directives (manually) when not using Emacs.

2.2.2 Version control (git) details

- CMItemplate uses git as a version control system with a central repositories on github.
  - CMItemplate uses the git-flow branching model
    - the principal development branch is `develop`
    - all new developments should be done on a `feature/` branch and, once ready, be branched into `develop`
  - Never touch the branch `master` – this is to be done by the maintainers.
    - the `master` branch is only for releases. There should never be any development done on `master`, nor any release preparations. The latter is done on `release/`, then the release is put onto `master`, and possibly necessary fixes are done on `hotfix/`.
  - Do not repeatedly branch feature branches into `develop` instead merge `develop` into your `feature/` branch.
  - General documentation work should always be made on `develop` (only)!
    - commit such doc-only updates as separate commits!
    - one can then merge these doc-only commits into `feature/` branches
  - **Never implement a change twice** manually. Implement it on the most appropriate branch, then merge it into whatever branch you want to have it.

2.2.3 Descriptions of source code files

Here one would provide a short introduction into the different code files, their intended content, and their interplay.

2.3 ToDo list

CHAPTER 3

Indices and tables

3.1 References

- genindex
- modindex
- search

Bibliography

[Küpper2020] Jochen Küpper, et al., Template for new CMI Python-software projects, GitHub (2020), https://github.com/CFEL-CMI/CMI-Python-project-template