
plico interferometer

Release 0.1.0

Arcetri Adaptive Optics

April 19, 2023

CONTENTS

1	Developer Documentation	3
2	API Reference	7
3	Indices and tables	11
	Python Module Index	13
	Index	15

Release

0.1.0

Date

April 19, 2023

plico_interferometer is an application to control interferometers.

DEVELOPER DOCUMENTATION

If you're interested in contributing to `plico_interferometer`, start here:

1.1 Development

1.1.1 Python version

- Python 3.6 or later

Builds on github automatically run all tests on Python versions 3.6, 3.7 and 3.8.

1.1.2 (human) Language

All functions, classes, and especially documentation shall be in English. Use a spell checker.

1.1.3 Coding style

- Style should follow PEP8: <https://www.python.org/dev/peps/pep-0008/>
- Documentation is generated automatically from docstrings, that should follow the numpydoc convention: <https://numpydoc.readthedocs.io/>

More details at https://github.com/ArcetriAdaptiveOptics/arte/blob/master/arte/code_convention.py

1.1.4 Developing guide

- Register on github.com and ask to be added to the list of arte's contributors
- Install git on your computer, and configure it with your name and contact email.
- Make sure that you have Python version 3.6+ installed.

Once this has been sorted out, here are the main steps:

Clone the repository

Clone the github repository on your computer:

- `git clone https://github.com/ArcetriAdaptiveOptics/plico_interferometer.git`
- `cd plico_interferometer`

Create a feature branch

Unless your change is trivial, it's best to create a branch to work independently:

```
git checkout -b mybranch
```

Develop

Make your changes and commit them into your branch. You can make as many commits as you want in your branch. Make sure that each commit has a readable description, because once developing is complete it will appear in the commit history for all to see, together with your name.

```
git commit -m "descriptive message"
```

Note: These commits are only visible on your computer and not to others, until you merge as described later. Moreover, since they are only kept on your local filesystem, there is no backup unless you provide one yourself.

Make a test

Each feature should have a unit test that verifies the functionality. Test files are in `arte/test` and follow the same structure as the main tree.

Execute tests with `pytest` and make sure that none are failing:

```
pytest
```

Keep your branch updated

Due to the distributed nature of git, it is possible that updates are made to the library while you are developing. It is recommended that from time to time you update your branch with these changes.

- `git checkout master` - switch to master branch
- `git pull` - pull all new changes from the github repository
- `git checkout mybranch` - switch back to your branch
- `git merge master` - merge changes from trunk into your branch

If someone has modified the same files as you, a conflict will arise at this point. You have to edit the files to a satisfactory resolution, and commit the result.

Publish

Once the feature is complete, it's time to make it available to others. Assuming that the previous steps are complete:

- all your changes are committed into your branch
- updates from master has been merged
- all tests pass

use this sequence to publish:

- `git checkout master` - switch to master branch
- `git merge mybranch` - merge changes from your branch into master
- `git push` - update master branch on github

If the branch have been kept updated, no conflicts should arise during merge.

After each push, github automatically runs all tests on a virtual machine. To see the result, click on the small icon next to the “Latest commit...” line on the right. Documentation at <https://arte.readthedocs.io/en/latest/index.htm> is also automatically updated.

Delete branch (optional)

Once the new feature has been merged to master, there is no need to keep the branch around:

- `git branch -d mybranch`

If you plan to further develop the feature, you can keep the branch and go on committing and merging as before.

API REFERENCE

The exact API of all functions and classes, as given by the docstrings. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

2.1 client

2.1.1 Module contents

2.1.2 Submodules

2.1.3 `plico_interferometer.client.interferometer_client` module

`class plico_interferometer.client.interferometer_client.InterferometerClient` (*rpcHandler*,
sockets)

Bases: *AbstractInterferometerClient*, *HackerableClient*, *ServerInfoClient*

snapshot (*prefix*, *timeout_in_sec=10.0*)

Parameters

prefix (*string*) – prefix to be prepended to the snapshot dict

Returns

snapshot – snapshot of the device to be used as FITS header

Return type

dict

status (*timeout_in_sec=10.0*)

Returns

status – status of the device

Return type

InterferometerStatus

wavefront (*how_many=1*, *timeout_in_sec=10.0*)

Parameters

how_many (*int* (*default=1*)) – return the average of *how_many* measurements.

Returns

wavefront – wavefront map in meters

Return type

array

2.1.4 `plico_interferometer.client.abstract_interferometer_client` module

class

`plico_interferometer.client.abstract_interferometer_client.AbstractInterferometerClient`

Bases: object

abstract `snapshot(prefix='MY_4D')`

Parameters

prefix (*string*) – prefix to be prepended to the snapshot dict

Returns

snapshot – snapshot of the device to be used as FITS header

Return type

dict

abstract `status()`

Returns

status – status of the device

Return type

InterferometerStatus

abstract `wavefront(how_many=1)`

Parameters

how_many (*int* (default=1)) – return the average of how_many measurements.

Returns

wavefront – wavefront map in meters

Return type

array

2.2 types

2.2.1 Submodules

2.2.2 `plico_interferometer.types.interferometer_status`

class `plico_interferometer.types.interferometer_status.InterferometerStatus(name)`

Bases: object

as_dict()

2.2.3 Module contents

2.3 utils

2.3.1 Submodules

2.3.2 `plico_interferometer.utils.timeout` module

```
class plico_interferometer.utils.timeout.Timeout
```

```
    Bases: object
```

```
    GETTER = 10.0
```

```
    SETTER = 10.0
```

2.3.3 Module contents

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

- plico_interferometer, 5
- plico_interferometer.client, 7
 - plico_interferometer.client.abstract_interferometer_client, 8
 - plico_interferometer.client.interferometer_client, 7
- plico_interferometer.types, 9
 - plico_interferometer.types.interferometer_status, 8
- plico_interferometer.utils, 9
 - plico_interferometer.utils.timeout, 9

INDEX

A

`AbstractInterferometerClient` (class in `plico_interferometer.client.abstract_interferometer_client`), 8
`as_dict()` (`plico_interferometer.types.interferometer_status` module), 8

G

GETTER (`plico_interferometer.utils.timeout.Timeout` attribute), 9

I

`InterferometerClient` (class in `plico_interferometer.client.interferometer_client`), 7
`InterferometerStatus` (class in `plico_interferometer.types.interferometer_status`), 8

M

module
 `plico_interferometer`, 5
 `plico_interferometer.client`, 7
 `plico_interferometer.client.abstract_interferometer_client`, 8
 `plico_interferometer.client.interferometer_client`, 7
 `plico_interferometer.types`, 9
 `plico_interferometer.types.interferometer_status`, 8
 `plico_interferometer.utils`, 9
 `plico_interferometer.utils.timeout`, 9

P

`plico_interferometer`
 module, 5
`plico_interferometer.client`
 module, 7
`plico_interferometer.client.abstract_interferometer_client`
 module, 8
`plico_interferometer.client.interferometer_client`
 module, 7

`plico_interferometer.types`

module, 9

`plico_interferometer.types.interferometer_status`
module, 8

`plico_interferometer.utils`
module, 9

`plico_interferometer.utils.timeout`
module, 9

S

SETTER (`plico_interferometer.utils.timeout.Timeout` attribute), 9

`snapshot()` (`plico_interferometer.client.abstract_interferometer_client.AbstractInterferometerClient` method), 8

`snapshot()` (`plico_interferometer.client.interferometer_client.InterferometerClient` method), 7

`status()` (`plico_interferometer.client.abstract_interferometer_client.AbstractInterferometerClient` method), 8

`status()` (`plico_interferometer.client.interferometer_client.InterferometerClient` method), 7

T

`Timeout` (class in `plico_interferometer.utils.timeout`), 9

W

`wavefront()` (`plico_interferometer.client.abstract_interferometer_client.AbstractInterferometerClient` method), 8

`wavefront()` (`plico_interferometer.client.interferometer_client.InterferometerClient` method), 7