acqpack Documentation

Release 1.2.0

Scott Longwell

Jun 07, 2021

Contents

1 AcqPack **3**

2 Installation **5**
2.1 Stable release 5
2.2 From sources 5

3 Usage **7**
3.1 Classes 7
3.2 Modules 13

4 Contributing **15**
4.1 Types of Contributions 15
4.2 Get Started! 16
4.3 Pull Request Guidelines 17
4.4 Tips 17

5 History **19**
5.1 0.1.0 (2017-07-03) 19

6 Indices and tables **21**

Python Module Index **23**

Index **25**

```
 Docs
Requirements
 Linux
 Windows
```

Code to perform acquisitions (experiment automation and hardware control). Should be combined with MMCorePy for microscope control.

-`config/` should be for computer scope settings
-`setup/` should be for experiment scope settings

```
from acqpack import ...
```

Contents:

CHAPTER 1

AcqPack

Docs
Requirements
Linux
Windows

Code to perform acquisitions (experiment automation and hardware control). Should be combined with MMCorePy for microscope control.

-`config/` should be for computer scope settings
-`setup/` should be for experiment scope settings

```
from acqpack import ...
```

CHAPTER 2

Installation

2.1 Stable release

To install acqpack, run this command in your terminal:

```
$ pip install acqpack
```

This is the preferred method to install acqpack, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for acqpack can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/fordycelab/acqpack
```

Or download the tarball:

```
$ curl  -OL https://github.com/fordycelab/acqpack/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use acqpack in a project:

```
import acqpack
```

3.1 Classes

class acqpack.**Motor**(*config_file*, *home=True*)

Low-level wrapper for the Lin Engineering (LE) CO-4118S-09. Config file must be defined.

The LE CO-4118S-09 has an integrated controller with a documented serial command-set. It lacks an encoder, and so relies on dead-reckoning for position. It does have an optical sensor that allows it to get a positional fix (home).

cmd(*cmd_string*, *block=True*)

Wraps core cmd_string with prefix and terminator specified in config, writes to serial, and returns response. Optionally blocks programmatic flow (default=True).

Parameters

- **cmd_string** – (str) core command (w/o prefix nor terminator)
- **block** – (bool) whether the command blocks program flow until action is complete

Returns (str) device response

is_busy()

Sends query command, then parses response to determine if motor is busy.

Returns (bool) true if motor is executing a command

set_velocity(*velocity*)

Checks requested velocity against the velocity limit, then sets motor velocity in usteps/sec.

Parameters **velocity** – (int) velocity

Returns (str) device response

halt()
Sends halt command to motor, which stops it from executing its current command. Note that many commands are sent in 'blocking' mode, so this function will likely not be called until the motor finishes executing its current command.

In the future, it may be nice to implement a 'waiting' scheme.

home()
Homes the motor until the optical sensor is triggered. Zero position is reset (motor gets positional fix).

Returns (str) device response

goto(*mm*, *block=True*)
Moves motor absolutely to the specified position.

Parameters

- **mm** – (float) desired absolute position [mm]
- **block** – (bool) whether the command blocks program flow until action is complete

Returns (str) device response

move_relative(*mm*)
Moves motor relatively by the specified number of mm.

Parameters **mm** – (float) desired relative movement [mm]

Returns (str) device response

where()
Retrieves motor's current position relative to zero-position (by dead reckoning).

Returns (tup) current position of the motor [mm]

exit()
Closes the device's serial connection.

class acqpack.**AsiController**(*config_file*, *init_xy=True*)
Low-level wrapper for the Applied Scientific Instrumentation (ASI) Controller. Config file must be defined.

This class can control both an XY-axis (MS-2000 stage) and a Z-axis (LS-50 linear stage). Since this hardware was taken from an Illumina GaIIx, it assumes the controller's serial command-set requires an OEM prefix of 1h (Z-axis) or 2h (XY-axes). Both stages have a linear-encoder.

Functions for Z-axis control are defined, but it is not initialized. If it is desired to be used, then a homing procedure needs to be defined in initialize().

cmd(*cmd_string*)
Wraps core cmd_string with terminator specified in config, writes to serial, and returns response.

Parameters **cmd_string** – (str) core command (w/o prefix nor terminator)

Returns (str) device response

halt()
Sends halt command to both axes, interrupting execution of their current commands. Note that many commands are sent in 'blocking' mode, so this function will likely not be called until the axes finish executing their current command.

In the future, it may be nice to implement a 'waiting' scheme.

exit()
Closes the device's serial connection.

cmd_xy(*cmd_string*, *block=True*)
Wraps core cmd_string with axes prefix (2h), passes to the cmd() function, and returns response. Optionally blocks programmatic flow (default=True).

Parameters

- **cmd_string** – (str) core command (w/o prefix nor terminator)
- **block** – (bool) whether the command blocks program flow until action is complete

Returns (str) device response

is_busy_xy()
Sends status command, then parses response to determine if XY-axes are busy.

Returns (bool) true if axes are executing a command

halt_xy()
Sends halt command to the XY-axes (stage), interrupting execution of its current command. Note that many commands are sent in 'blocking' mode, so this function will likely not be called until the axes finish executing their current command.

In the future, it may be nice to implement a 'waiting' scheme.

zero_xy(*x_dir=1*, *y_dir=1*)
Sets the origin (zeros) at current location. If 'x_dir' and 'y_dir' are specified, will seek hardware limit (hall-effect stops) before zeroing. 'x_dir' and 'y_dir' represent whether to max (+1) or min (-1) each axis.

Parameters

- **x_dir** – (int) -1 to min, +1 to max
- **y_dir** – (int) -1 to min, +1 to max

Returns (str) device response

home_xy()
Moves XY-axes to origin (0,0)

where_xy()
Retrieves XY-axes' current position relative to zero point (w/ linear encoder).

Returns (tup) current X and Y position [mm]

goto_xy(*x_mm*, *y_mm*)
Moves XY-axes absolutely to the specified position.

Parameters

- **x_mm** – (float) desired absolute X position [mm]
- **y_mm** – (float) desired absolute Y position [mm]

Returns (str) device response

move_relative_xy(*x_mm*, *y_mm*)
Moves XY-axes relatively by the specified number of mm.

Parameters

- **x_mm** – (float) desired relative movement [mm]
- **y_mm** – (float) desired relative movement [mm]

Returns (str) device response

cmd_z(*cmd_string*, *block=True*)
Wraps core cmd_string with axis prefix (1h), passes to the cmd() function, and returns response. Optionally blocks programmatic flow (default=True).

Parameters

- **cmd_string** – (str) core command (w/o prefix nor terminator)
- **block** – (bool) whether the command blocks program flow until action is complete

Returns (str) device response

is_busy_z()
Sends status command, then parses response to determine if Z-axis is busy.

Returns (bool) true if axis is executing a command

halt_z()
Sends halt command to the Z-axis (linear motor), interrupting execution of its current command. Note that many commands are sent in 'blocking' mode, so this function will likely not be called until the axes finish executing their current command.

In the future, it may be nice to implement a 'waiting' scheme.

home_z()
Moves Z-axis to 0.

where_z()
Retrieves Z-axis' current position relative to zero point (w/ linear encoder).

Returns (tup) current Z position [mm]

goto_z(*z_mm*)
Moves Z-axis absolutely to the specified position.

Parameters **z_mm** – (float) desired absolute Z position [mm]

Returns (str) device response

move_relative_z(*z_mm*)
Moves Z-axis relatively by the specified number of mm.

Parameters **z_mm** – (float) desired relative movement [mm]

Returns (str) device response

class acqpack.**Autosampler**(*z*, *xy*)
A high-level wrapper that coordinates XY and Z axes to create an autosampler. Incorporates a deck.

add_frame(*name, trans=array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), position_table=None*)
Adds coordinate frame. Frame requires affine transform to hardware coordinates; position_table optional.

Parameters

- **name** – (str) the name to be given to the frame (e.g. hardware)
- **trans** – (np.ndarray <- str) xyzw affine transform matrix; if string, tries to load delimited file
- **position_table** – (None | pd.DataFrame <- str) position_table; if string, tries to load delimited file

add_plate(*name*, *filepath*, *ref_frame='deck'*)
TODO: UNDER DEVELOPMENT

where (*frame=None*)
Retrieves current hardware (x,y,z). If frame is specified, transforms hardware coordinates into frame's coordinates.

Parameters **frame** – (str) name of frame to specify transform (optional)

Returns (tup) current position

home ()
Homes Z axis, then XY axes.

goto (*frame*, *lookup_columns*, *lookup_values*, *zh_travel=0*)
Finds lookup_values in lookup_columns of frame's position_list; retrieves corresponding X,Y,Z. Transforms X,Y,Z to hardware X,Y,Z by frame's transform. Moves to hardware X,Y,Z, taking into account zh_travel.

Parameters

- **frame** – (str) frame that specifies position_list and transform
- **lookup_columns** – (str | list) column(s) to search in position_table
- **lookup_values** – (val | list) values(s) to find in lookup_columns
- **zh_travel** – (float) hardware height at which to travel

exit ()
Send exit command to XY and Z

class acqpack.**FractionCollector**(*xy*)
A high-level wrapper around an XY stage.

add_frame (*name, trans=array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]]), position_table=None*)
Adds coordinate frame. Frame requires affine transform to hardware coordinates; position_table optional.

Parameters

- **name** – (str) the name to be given to the frame (e.g. hardware)
- **trans** – (np.ndarray <- str) xyw affine transform matrix; if string, tries to load delimited file
- **position_table** – (None | pd.DataFrame <- str) position_table; if string, tries to load delimited file

where (*frame=None*)
Retrieves current hardware (x,y). If frame is specified, transforms hardware coordinates into frame's coordinates.

Parameters **frame** – (str) name of frame to specify transform (optional)

Returns (tup) current position

home ()
Homes XY axes.

goto (*frame*, *lookup_columns*, *lookup_values*)
Finds lookup_values in lookup_columns of frame's position_list; retrieves corresponding X,Y Transforms X,Y to hardware X,Y by frame's transform. Moves to hardware X,Y.

Parameters

- **frame** – (str) frame that specifies position_list and transform
- **lookup_columns** – (str | list) column(s) to search in position_table

- **lookup_values** – (val | list) values(s) to find in lookup_columns

exit()
Send exit command to XY.

class acqpack.**Manifold**(*ip_address*, *valvemap_path*, *read_offset=512*)
Provides a wrapper for the manifold, which is controlled by the Wago nModbus

load_valvemap(*valvemap_path*)
Stores valvemap. To work with open/close, valvemap should have one column named 'valve'.

Parameters **valvemap_path** – (str) path to valvemap

read_valve(*valve_num*)
Reads the state of the register associated with the specified valve.

Parameters **valve_num** – (int) register number to read

Returns () state of the register (True: depressurized, False: pressurized)

pressurize(*valve_num*)
Pressurizes valve at the specified register.

Parameters **valve_num** – (int) valve to pressurize

depressurize(*valve_num*)
Depressurizes valve at the specified register.

Parameters **valve_num** – (int) valve to depressurize

close(*lookup_cols*, *lookup_vals*)
Finds lookup_vals in lookup_cols of valvemap; retrieves corresponding valve_num. Closes valve_num.

Parameters

- **lookup_cols** – (str | list) column(s) to search in valvemap
- **lookup_vals** – (val | list) value(s) to find in lookup_cols

open(*lookup_cols*, *lookup_vals*)
Finds lookup_vals in lookup_cols of valvemap; retrieves corresponding valve_num. Opens valve_num.

Parameters

- **lookup_cols** – (str | list) column(s) to search in valvemap
- **lookup_vals** – (val | list) value(s) to find in lookup_cols

exit()
Closes the device's serial connection.

class acqpack.**Mfcs**(*config_file*, *chanmap_path*)
Class to control the MFCS-EZ.

detect()
Detects up to 8 connected MFCS devices; returns serial numbers of connected devices.

Returns (list) detected MFCS serial numbers as ints

connect()
Initializes the MFCS. Makes connection, checks status, and sets the PID alpha parameter of all channels to 2.

status()
Gets and returns status of the MFCS. 0: 'MFCS is reset - press "Play"' 1: 'normal' 2: 'overpressure' 3: 'need to rearm'

Returns (tup) status int [0-3], status string

pid(*chan*, *a*)
Sets alpha parameter of the PID controller for the given channel. Lower values of alpha (1-2) are typically more stable at lower pressures, but take slightly longer to equilibrate.

For some reason, the python kernel would crash when 'channel' and 'alpha' were used as keywords. C-types...

Parameters

- **chan** – (int) channel [1-4] to set; 0 sets for all channels
- **a** – (int) desired alpha value for PID

load_chanmap(*chanmap_path*)
Stores channel map.

Parameters **chanmap_path** – (str) path to chanmap

exit()
Safely closes the MFCS. First closes device connection, then releases the DLL.

set(*lookup_cols*, *lookup_vals*, *pressure=0.0*)
Sets pressure of specified channel.

Parameters

- **lookup_cols** – (str | list) column(s) to search in chanmap
- **lookup_vals** – (val | list) value(s) to find in lookup_cols
- **pressure** – (float) desired pressure; units specified in config file

read(*lookup_cols*, *lookup_vals*)
Reads current pressure of the channel.

Parameters

- **lookup_cols** – (str | list) column(s) to search in chanmap
- **lookup_vals** – (val | list) value(s) to find in lookup_cols

Returns (float) current pressure; units specified in config file

3.2 Modules

acqpack.gui.**imshow**(*img*, *name='Image'*, *mode='cv2'*)

acqpack.gui.**snap**(*core*, *mode='mpl'*)

acqpack.gui.**video**(*core*, *loop_pause=0.15*)

acqpack.gui.**grid**(*core*, *loop_pause=0.15*)

acqpack.gui.**manifold_control**(*manifold*, *button_col='name'*, *trim_to_valvemap=True*)

acqpack.gui.**stage_control**(*stage*)

acqpack.utils.**read_delim**(*filepath*)
Reads delimited file (auto-detects delimiter + header). Returns list.

Parameters **filepath** – (str) location of delimited file

Returns (list) list of records w/o header

acqpack.utils.**read_delim_pd**(*filepath*)
Reads delimited file (auto-detects delimiter + header). Returns pandas DataFrame.

Parameters **filepath** – (str) location of delimited file

Returns (DataFrame)

acqpack.utils.**lookup**(*table*, *lookup_cols*, *lookup_vals*, *output_cols=None*, *output_recs=None*)
Looks up records where lookup_cols == lookup_vals. Optionally returns only specified output_cols and/or specified output_recs.

Parameters

- **table** – (DataFrame) the pandas DataFrame to use as a lookup table
- **lookup_cols** – (str | list)
- **lookup_vals** – (val | list)
- **output_cols** –
- **output_recs** –

Returns

acqpack.utils.**generate_position_table**(*num_rc*, *space_rc*, *offset=(0.0, 0.0, 0.0)*, *to_clipboard=False*)
Generates a position table for a plate. Assumes that 'x' and 'c' are aligned and that 'y' and 'r' are aligned. These axes can be reflected by negating the corresponding 'space_rc'; translations can be applied via 'offset'. All entries are indexed by 'n' (newspaper order) and 's' (serpentine order). Other columns may be added as needed, but Autosampler.goto() requires 'x', 'y', and 'z' to function properly.

Parameters

- **num_rc** – (tup) number of rows and columns (num_rows, num_cols)
- **space_rc** – (tup) spacing for rows and columns [mm] (spacing_rows, spacing_cols)
- **offset** – (tup) 3-tuple of floats to be added to x,y,z [mm]
- **to_clipboard** – (bool) whether to copy the position_table to the OS clipboard

Returns (DataFrame)

acqpack.utils.**spacing**(*num_rc*, *p1*, *p2*)

acqpack.utils.**load_mm_positionlist**(*filepath*)
Takes a MicroManager position list and converts it to a pandas DataFrame. Will load z-coordinates if available.

Parameters **filepath** – (str)

Returns (DataFrame) position list with headers = "r, c, name, x, y, [z]"

acqpack.utils.**generate_grid**(*c0*, *c1*, *l_img*, *p*)
Based on two points, creates a 2D-acquisition grid similar to what MicroManager would produce.

Parameters

- **c0** – (arr) first point; numpy 1d array of len 2
- **c1** – (arr) second point; numpy 1d array of len 2
- **l_img** – (float)
- **p** – (float) desired percent overlap

Returns (DataFrame) position_list in the same format as load_mm_positionlist

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 https://github.com/fordycelab/acqpack/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" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

acqpack could always use more documentation, whether as part of the official acqpack 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 https://github.com/fordycelab/acqpack/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 *acqpack* for local development.

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

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

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

```
$ mkvirtualenv acqpack
$ cd acqpack/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b 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:

```
$ flake8 acqpack tests
$ python setup.py test or py.test
$ 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 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 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/fordycelab/acqpack/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_acqpack
```

CHAPTER 5

History

5.1 0.1.0 (2017-07-03)

- First release on PyPI.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

acqpack.gui, 13
acqpack.utils, 13

Index

A

acqpack.gui (module), 13
acqpack.utils (module), 13
add_frame() (acqpack.Autosampler method), 10
add_frame() (acqpack.FractionCollector method), 11
add_plate() (acqpack.Autosampler method), 10
AsiController (class in acqpack), 8
Autosampler (class in acqpack), 10

C

close() (acqpack.Manifold method), 12
cmd() (acqpack.AsiController method), 8
cmd() (acqpack.Motor method), 7
cmd_xy() (acqpack.AsiController method), 8
cmd_z() (acqpack.AsiController method), 9
connect() (acqpack.Mfcs method), 12

D

depressurize() (acqpack.Manifold method), 12
detect() (acqpack.Mfcs method), 12

E

exit() (acqpack.AsiController method), 8
exit() (acqpack.Autosampler method), 11
exit() (acqpack.FractionCollector method), 12
exit() (acqpack.Manifold method), 12
exit() (acqpack.Mfcs method), 13
exit() (acqpack.Motor method), 8

F

FractionCollector (class in acqpack), 11

G

generate_grid() (in module acqpack.utils), 14
generate_position_table() (in module acqpack.utils), 14
goto() (acqpack.Autosampler method), 11
goto() (acqpack.FractionCollector method), 11
goto() (acqpack.Motor method), 8
goto_xy() (acqpack.AsiController method), 9
goto_z() (acqpack.AsiController method), 10
grid() (in module acqpack.gui), 13

H

halt() (acqpack.AsiController method), 8
halt() (acqpack.Motor method), 7
halt_xy() (acqpack.AsiController method), 9
halt_z() (acqpack.AsiController method), 10
home() (acqpack.Autosampler method), 11
home() (acqpack.FractionCollector method), 11
home() (acqpack.Motor method), 8
home_xy() (acqpack.AsiController method), 9
home_z() (acqpack.AsiController method), 10

I

imshow() (in module acqpack.gui), 13
is_busy() (acqpack.Motor method), 7
is_busy_xy() (acqpack.AsiController method), 9
is_busy_z() (acqpack.AsiController method), 10

L

load_chanmap() (acqpack.Mfcs method), 13
load_mm_positionlist() (in module acqpack.utils), 14
load_valvemap() (acqpack.Manifold method), 12
lookup() (in module acqpack.utils), 14

M

Manifold (class in acqpack), 12
manifold_control() (in module acqpack.gui), 13
Mfcs (class in acqpack), 12
Motor (class in acqpack), 7
move_relative() (acqpack.Motor method), 8
move_relative_xy() (acqpack.AsiController method), 9
move_relative_z() (acqpack.AsiController method), 10

O

open() (acqpack.Manifold method), 12

P

pid() (acqpack.Mfcs method), 13

pressurize() (acqpack.Manifold method), 12

R

read() (acqpack.Mfcs method), 13
read_delim() (in module acqpack.utils), 13
read_delim_pd() (in module acqpack.utils), 13
read_valve() (acqpack.Manifold method), 12

S

set() (acqpack.Mfcs method), 13
set_velocity() (acqpack.Motor method), 7
snap() (in module acqpack.gui), 13
spacing() (in module acqpack.utils), 14
stage_control() (in module acqpack.gui), 13
status() (acqpack.Mfcs method), 12

V

video() (in module acqpack.gui), 13

W

where() (acqpack.Autosampler method), 10
where() (acqpack.FractionCollector method), 11
where() (acqpack.Motor method), 8
where_xy() (acqpack.AsiController method), 9
where_z() (acqpack.AsiController method), 10

Z

zero_xy() (acqpack.AsiController method), 9