# batlab-software-python Documentation

*Release 0.5.36*

Lexcelon, LLC

Apr 16, 2019

# Contents:

CHAPTER 1

# Batlab library for Python

[master] [develop]

batlab-software-python is a Python library and example command line script to interact with a pool of Batlabs over USB. This tool is designed for hobbyists and more advanced users who would like to incorporate the Batlab hardware in their own cell testing workflow or environment.

## 1.1 Requirements

Python >=3.4 is supported by this module.

Python 2 is not supported.

## 1.2 Installation

To install the latest release you can use pip:

```
$ pip install batlab
```

To upgrade, you can run either of:

```
$ pip install batlab --upgrade
$ pip install batlab -U
```

## 1.3 Documentation

Documentation for this library is hosted at Read the Docs. Please go there to find this README as well as the available documentation for classes and methods within this library.

## 1.4 Usage

The library can be imported into your own programs, or you may use the provided example Batlab Utility Script.

### 1.4.1 Batlab Utility Script

The Batlab Utility Script allows users to perform basic interactions with a pool of connected Batlab units through a simple command-line interface.

To run the script, make sure the `batlab` package is installed and then run:

```
$ batlabutil
```

Type `help` to display the list of commands in the script and how to use them. The intention for the script is to serve as an example for users to write their own battery cell test software using the Batlab library.

## 1.5 Contributing

When contributing to this repository, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository.

### 1.5.1 Git branching model

We follow the development model described here. Anything in the `master` branch is considered production. Most work happens in the `develop` branch or in a feature branch that is merged into `develop` before being merged into `master`.

### 1.5.2 Documenting

Please document code (notably functions and classes) using doc strings according to the Google Python Style Guide standards. This will ensure that your notes are automatically picked up and included in the generated documentation.

### 1.5.3 Running tests

To run the unit tests, from the root directory run:

```
$ python setup.py test
```

### 1.5.4 Deployment

This library is deployed to PyPi. Builds are generated with Travis CI with each pushed commit. When a new tag is pushed or merged into `master`, that build is automatically deployed to end users through PyPi.

To publish a new version, the workflow might look like this. First make your changes:

```
$ git commit -am "some changes to the develop branch"
```

Then, update `setup.py` and `docs/source/conf.py` with the new version number (in this case we will use the example 0.100.56). Make a commit with these changes and push it to GitHub:

```
$ git commit -am "rev version number to 0.100.56"
$ git push origin develop
```

Then you must wait a couple of minutes to make sure the build passes on Travis. If the build fails, you will not be able to merge the commit into `master`. Once the build passes, you can merge into master, create a tagged release and push. When merging, use `--no-ff` to preserve the commit and branching history:

```
$ git checkout master
$ git merge --no-ff develop
$ git tag v0.100.56
$ git push origin master
$ git push --tags
```

Changes should automatically roll out to PyPi, and any documentation included in your code will automatically roll out to Read the Docs.

## 1.6 License

## 1.7 Acknowledgements

Thank you to our backers on Kickstarter who made this project possible.

Documentation created with guide from Sam Nicholls.

# CHAPTER 2

# Indices and tables

- genindex
- modindex
- search