the labscript suite

Release 3.0.1.dev0+gfba5b50.d20200627

labscript suite contributors

Jun 27, 2020

DOCUMENTATION

CHAPTER

ONE

EXPERIMENT CONTROL AND AUTOMATION SYSTEM

The *labscript suite* is a powerful and extensible framework for experiment composition, control, execution, and analysis. Developed for quantum science and quantum engineering; deployable in laboratory and in-field devices. Also applicable to optics, microscopy, materials engineering, biophysics, and any application predicated on the repetition of parameterised, hardware-timed experiments.

1.1 Features

- Flexible and automated oversight of heterogeneous hardware.
- The most mature and widely used open-source control system in quantum science.
- Multiple analysis-based feedback modes.
- Extensible plugin architecture (e.g. machine learning online optimisation).
- Readily integrates with other software, including image acquisition, analysis, and even other control systems.
- Compose experiments as human-readable Python code, leveraging modularity, revision control and re-use.
- Dynamic visualisation of experiment composition and results.
- Remote operation: different modules can run on physically separate hosts / single modules can be run on multiple hosts (including hardware supervisor, blacs).
- Auto-generating user-interfaces.
- High-level scripting: user-interface interaction can be programatically synthesised.

1.2 Citing the *labscript suite*

If you use the *labscript suite* to control your experiment or perform analysis, please cite one or more of the following publications:

```
@phdthesis{starkey_phd_2019,
   title = {State-dependent forces in cold quantum gases},
   author = {Starkey, P. T.},
   year = {2019},
   url = {https://doi.org/10.26180/5d1db8ffe29ef},
   doi = {10.26180/5d1db8ffe29ef},
   school = {Monash University},
}
```

```
@phdthesis{billington_phd_2018,
   title = {State-dependent forces in cold quantum gases},
   author = {Billington, C. J.},
   year = {2018},
   url = {https://doi.org/10.26180/5bd68acaf0696},
   doi = {10.26180/5bd68acaf0696},
   school = {Monash University},
}
```

```
@article{labscript_2013,
   author = {Starkey, P. T. and Billington, C. J. and Johnstone, S. P. and
             Jasperse, M. and Helmerson, K. and Turner, L. D. and Anderson, R. P.},
   title = {A scripted control system for autonomous hardware-timed experiments},
   journal = {Review of Scientific Instruments},
   volume = {84},
   number = {8},
   pages = {085111},
   year = {2013},
   doi = {10.1063/1.4817213},
   url = {https://doi.org/10.1063/1.4817213},
   eprint = {https://doi.org/10.1063/1.4817213}
}
```

1.2.1 Installing the *labscript suite*

We're excited to announce that accompanying the recent migration of the codebase from BitBucket to GitHub, *labscript suite* components are now distributed as Python packages on PyPI and Anaconda Cloud.

This makes it far easier to get started using the *labscript suite*, as you no longer require a Mercurial or Git installation (or any knowledge of version control software); components can be installed and upgraded using:

- `pip`: the standard package manager common to all Python distributions; or
- `conda`: a binary package and environment manager, part of the Anaconda Python distribution.

Setting up a Python environment

We recommend installing the *labscript suite* (regular or developer mode) in a virtual environment. This helps sandbox the codebase without interfering with (or being interfered with) your system Python installation, or Python environments used for other purposes. Below we outline how to create and activate a virtual environment for Anaconda Python and other CPython distributions (which we call 'Regular' Python here).

Anaconda Python

Anaconda Python includes a virtual environment manager as part of the `conda` executable. Here's an example (on Windows):

Note: Make sure you have opened the Anaconda Prompt on Windows (available from the Start menu). `conda` is not available from the standard terminal by default. Launching 'Anaconda Prompt' will activate the `base` conda environment.

Warning: We do not recommend using the `base` conda environment for any project (*labscript suite* or otherwise). Working in a separate conda environment ensures any package resolution or update errors (however unlikely) are limited to that environment, and `base` is not compromised.

Quickstart

```
(base) C:\> conda create -n py38 python=3.8
(base) C:\> conda activate py38
(py38) C:\>
```

Once activated, the name of the virtual environment (in this case, `py38`) will prefix the command line.

Detailed Instructions

1. Create a virtual (conda) environment. Here we name it `py38` and ask conda to use Python 3.8 within the virtual environment (name and Python version are variable but these are conventional choices):

   ```
   (base) C:\> conda create -n py38 python=3.8
   ```

2. Activate the virtual (conda) environment:

   ```
   (base) C:\> conda activate py38
   (py38) C:\>
   ```

Regular Python

There are a number of ways to configure a virtual environment. If you are unfamiliar with doing so, we recommend using the venv module, part of the Python Standard Library. Here's an example (on Windows):

Quickstart

```
C:\Users\wkheisenberg> mkdir labscript-suite
C:\Users\wkheisenberg> cd labscript-suite
C:\Users\wkheisenberg\labscript-suite> python -m venv .venv
C:\Users\wkheisenberg\labscript-suite> .venv\Scripts\activate
(.venv) C:\Users\wkheisenberg\labscript-suite> python -m pip install --upgrade pip␣
↪setuptools wheel
```

Once activated, the name of the virtual environment (in this case, `.venv`) will prefix the command line.

Detailed Instructions

1. From a new terminal, create a directory for the virtual environment and enter it. Here we use a directory named `labscript-suite` in the user's home directory, also the location of the labscript suite profile directory. You need create the virtual environment here, but it is a convenient choice.

```
C:\Users\wkheisenberg> mkdir labscript-suite
C:\Users\wkheisenberg> cd labscript-suite
```

2. Create a virtual environment. Here we name it `.venv`, located inside the labscript suite profile directory.

```
C:\Users\wkheisenberg\labscript-suite> python -m venv .venv
```

3. Activate the virtual environment:

```
C:\Users\wkheisenberg\labscript-suite> .venv\Scripts\activate
```

Note: This step is OS specific, e.g. on Linux it's `source .venv/bin/activate`.

---

4. Update the Python package installer and other installation packages of your virtual environment.

```
(.venv) C:\Users\wkheisenberg\labscript-suite> python -m pip install --
↪upgrade pip setuptools wheel
```

Choosing an installation method

Once you have a virtual environment up and running, choose from one of the following 4 installation methods:

1. *Regular installation (Python Package Index)*;
2. *Regular installation (Anaconda Cloud)*;
3. *Developer installation (Python Package Index)*; or
4. *Developer installation (Anaconda Cloud)*.

Regular installation (Python Package Index)

In this example, we will use an existing virtual environment named `.venv` located in `C:\Users\wkheisenberg\labscript-suite`. Skip the first two lines/steps if continuing on from the instructions to *set up this environment*.

Quick start

```
C:\Users\wkheisenberg\labscript-suite> .venv\Scripts\activate
(.venv) C:\Users\wkheisenberg\labscript-suite> python -m pip install --upgrade pip␣
↪setuptools wheel
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install labscript-suite
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install PyQt5
(.venv) C:\Users\wkheisenberg\labscript-suite> labscript-profile-create
(.venv) C:\Users\wkheisenberg\labscript-suite> desktop-app install blacs lyse␣
↪runmanager runviewer
```

Detailed instructions

1. Activate the virtual environment (this step is OS specific, e.g. on Linux it's `source .venv/bin/activate`).

```
C:\Users\wkheisenberg\labscript-suite> .venv\Scripts\activate
```

2. Update the Python package installer and other installation packages of your virtual environment.

```
(.venv) C:\Users\wkheisenberg\labscript-suite> python -m pip install --
↪upgrade pip setuptools wheel
```

3. Install the meta-package (`labscript-suite`) from PyPI. This will install blacs, labscript, labscript-devices, labscript-utils, lyse, runmanager, runviewer, and all dependencies:

```
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install labscript-suite
```

4. Install PyQt5, the bindings to the GUI toolkit (not installed above for licensing reasons):

```
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install PyQt5
```

5. Create (or populate) a profile directory in your home directory (the location of user data; see *Recent changes to the labscript suite*):

```
(.venv) C:\Users\wkheisenberg\labscript-suite> labscript-profile-create
```

6. (Optional) Create shortcuts for the GUI applications (blacs, lyse, runmanager, and runviewer) and place them in the start-menu (or non-Windows OS equivalent).

```
(.venv) C:\Users\wkheisenberg\labscript-suite> desktop-app install blacs lyse␣
↪runmanager runviewer
```

   These will be named, e.g. 'runmanager – the labcript suite' which when clicked on will:

   - Launch the application without a terminal window, using the virtual environment the above command was called in.
   - Display the application with an application-specific shortcut in the taskbar (which can be pinned, like any other desktop application).

Note: Virtual environments named anything other than `.venv` will be included in the name of the shortcut, e.g. 'runmanager – the labscript suite (py38)' for a virtual environment named `py38`.

Alternatively, you can launch the applications from a terminal, e.g.

```
(.venv) C:\> runmanager
```

This will print debugging information to the console.

To launch the applications detached from the console, suffix the application name with `-gui`, e.g.

```
(.venv) C:\> runmanager-gui
```

Note: You must have activated the virtual environment in which the *labscript suite* was installed to use these commands.

Updating a regular installation

Individual components of the labscript suite can be updated using the `--upgrade` (`-U`) flag of `pip`. For example:

```
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install -U runmanager
```

To upgrade to a pre-release version, you can use the `--pre` (pre-relase) flag:

```
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install -U --pre runmanager
```

If updating multiple components, use a single `pip install` command to assist dependency resolution:

```
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install -U labscript lyse␣
↪runmanager
```

You can also update (or downgrade) to a specific version:

```
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install runmanager==2.5.0
```

Regular installation (Anaconda Cloud)

In this example, we will use an existing conda environment named `py38`. Skip the first line/step if continuing on from the instructions to *set up this environment*.

Quick start

```
(base) C:\> conda activate py38
(base) C:\> conda config --env --add channels labscript-suite
(py38) C:\> conda install labscript-suite pyqt
(py38) C:\> labscript-profile-create
(py38) C:\> desktop-app install blacs lyse runmanager runviewer
```

Detailed instructions

1. Activate the conda environment from the Anaconda Prompt.

   ```
   (base) C:\> conda activate py38
   ```

2. Add the `labscript-suite` channel on Anaconda Cloud to the current conda environment:

   ```
   (py38) C:\> conda config --env --add channels labscript-suite
   ```

3. Install the meta-package (`labscript-suite`) and bindings to the GUI toolkit (`pyqt`) from Anaconda Cloud. This will install blacs, labscript, labscript-devices, labscript-utils, lyse, runmanager, runviewer, and all dependencies:

   ```
   (py38) C:\> conda install labscript-suite pyqt
   ```

4. Create a profile directory in your home directory (the location of user data; see *Recent changes to the labscript suite*):

```
(py38) C:\> labscript-profile-create
```

5. (Optional) Create shortcuts for the GUI applications (blacs, lyse, runmanager, and runviewer) and place them in the start-menu (or non-Windows OS equivalent).

   ```
   (py38) C:\> desktop-app install blacs lyse runmanager runviewer
   ```

   These will be named, e.g. 'runmanager – the labcript suite (py38)' which when clicked on will:

   - Launch the application without a terminal window, using the virtual environment the above command was called in.
   - Display the application with an application-specific shortcut in the taskbar (which can be pinned, like any other desktop application).

Note: Conda environments named anything other than `base` will be included in the name of the shortcut, e.g. 'runmanager – the labscript suite (py38)' for a conda environment named `py38`.

Alternatively, you can launch the applications from the Anaconda Prompt in the , e.g.

```
(py38) C:\> runmanager
```

This will print debugging information to the console.

To launch the applications detached from the console, suffix the application name with `-gui`, e.g.

```
(.venv) C:\> runmanager-gui
```

Note:

- You must have activated the conda environment in which the *labscript suite* was installed to use these commands.
- For the `-gui` entry points to function in Anaconda Python, Step 5 (above) must be completed.

Updating a regular installation

Individual components of the labscript suite can be updated using the `conda update` command. For example:

```
(py38) C:\> conda update -c labscript-suite runmanager
```

To upgrade to a pre-release version, you can use the test label:

```
(py38) C:\> conda upadte -c labscript-suite/label/test runmanager
```

If updating multiple components, use a single `conda update` command to assist dependency resolution:

```
(py38) C:\> conda update -c labscript-suite labscript lyse runmanager
```

You can also update (or downgrade) to a specific version:

```
(py38) C:\> conda update runmanager==2.5.0
```

Developer installation (Python Package Index)

Developer installations are useful for those who want to customise the *labscript suite*.

Note: You need not fork, clone, and install editable versions of all *labscript suite* repositories to customise your installation and/or contribute changes back to the base repositories. For example, if you only want to develop custom labscript device drivers, you might only fork and clone the labscript-devices repository. Moreover, there is now an option to write and use custom labscript device drivers outside of the labscript-devices installation directory.

Quick start

```
C:\Users\wkheisenberg\labscript-suite> .venv\Scripts\activate
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install \
        --src . -e git+https://github.com/wkheisenberg/blacs#egg=blacs \
        --src . -e git+https://github.com/wkheisenberg/labscript#egg=labscript \
        --src . -e git+https://github.com/wkheisenberg/labscript-devices
↪#egg=labscript-devices \
        --src . -e git+https://github.com/wkheisenberg/labscript-utils#egg=labscript-
↪utils \
        --src . -e git+https://github.com/wkheisenberg/runmanager#egg=runmanager \
        --src . -e git+https://github.com/wkheisenberg/runviewer#egg=runviewer \
        --src . -e git+https://github.com/wkheisenberg/lyse#egg=lyse
(.venv) C:\Users\wkheisenberg\labscript-suite> pip install PyQt5
(.venv) C:\Users\wkheisenberg\labscript-suite> labscript-profile-create
(.venv) C:\Users\wkheisenberg\labscript-suite> desktop-app install blacs lyse␣
↪runmanager runviewer
```

Detailed instructions

1. Fork the labscript-suite repositories you want to develop using the GitHub online interface. Below we will include all repositories (except the labscript-suite metapackage).
2. Use `pip` to both clone these forks locally and install them into your environment. In this example (on Windows), the forks are owned by the (non-existent) GitHub user wkheisenberg.

   ```
   C:\Users\wkheisenberg\labscript-suite> pip install ^
   --src . -e git+https://github.com/wkheisenberg/blacs#egg=blacs ^
   --src . -e git+https://github.com/wkheisenberg/labscript#egg=labscript ^
   --src . -e git+https://github.com/wkheisenberg/labscript-devices#egg=labscript-
    ↪devices ^
   --src . -e git+https://github.com/wkheisenberg/labscript-utils#egg=labscript-
    ↪utils ^
   --src . -e git+https://github.com/wkheisenberg/runmanager#egg=runmanager ^
   --src . -e git+https://github.com/wkheisenberg/runviewer#egg=runviewer ^
   --src . -e git+https://github.com/wkheisenberg/lyse#egg=lyse
   ```

   Note:

   - This will set your forked repository(ies) to be the 'origin' remote.
   - On Linux / macOS the line continuation character is \ rather than ^.

Alternatively, manually clone the repositories using `git clone` and then install them using `pip` by running the following from the common parent directory:

```
C:\Users\wkheisenberg\labscript-suite> pip install -e blacs -e labscript \
    -e labscript-devices -e labscript-utils -e lyse -e runmanager -e runviewer
```

For a single package, this would look like:

```
C:\Users\wkheisenberg\labscript-suite> git clone https://github.com/wkheisenberg/
↪runmanager.git
C:\Users\wkheisenberg\labscript-suite> pip install -e runmanager
```

3. For each repository, set the upstream remote to the base labscript-suite repository:

```
C:\Users\wkheisenberg\labscript-suite> cd blacs
C:\Users\wkheisenberg\labscript-suite> git remote add upstream https://github.com/
↪labscript-suite/blacs.git
C:\Users\wkheisenberg\labscript-suite> cd ..
```

Repeat for the other repositories.

4. Continue from step 4 (install PyQt5) in the *Regular installation (Python Package Index)* instructions.

Updating a developer installation

This assumes you have already completed the developer installation above and have:

- Forked a *labscript suite* repository on GitHub;
- Cloned the repository;
- Set your fork to be the 'origin' remote; and
- Set the labscript-suite base repository to be the 'upstream' remote.

1. Use one of the following to keep your repository (and feature branches) up-to-date:

   Fetch changes, and merge with your local master branch.

   ```
   > git checkout master
   > git fetch upstream master --tags
   > git merge upstream/master
   ```

   Or using Git Pull:

   ```
   > git checkout master
   > git pull upstream master --tags
   ```

   Or using hub sync command-line extension (does not require current local working branch to be master):

   ```
   > hub sync
   ```

2. Update your feature branches by merging them with master or rebasing them to master:

   ```
   > git checkout your-feature-name
   > git merge master <OR> git rebase master --autostash
   ```

3. Update your fork by pushing any changes resulting from steps 1–2 and/or subsequent local development:

```
> git checkout master
> git push origin master --tags
> git checkout your-feature-name
> git push origin your-feature-name master
```

Note: If the feature branch has not yet been created on your fork, you need to include `-u` above, i.e.

```
> git push -u origin your-feature-name
```

4. Checkout the commit you want to install. This might be a specific release version (which can be specified by tag):

   ```
   > git checkout v0.3.2
   ```

   or using the commit SHA:

   ```
   > git checkout 59651b5
   ```

5. (Optional) Update the package using (from within the root of a repository):

   ```
   > pip install -e .
   ```

   As the installations are in editable mode and the version is being introspected at runtime, this step is not always necessary, but is required for any change requiring setup.py to be run to take effect, e.g. dependency changes, console entry points, etc.

Developer installation (Anaconda Cloud)

Developer installations are useful for those who want to customise the *labscript suite*.

Note: You need not fork, clone, and install editable versions of all *labscript suite* repositories to customise your installation and/or contribute changes back to the base repositories. For example, if you only want to develop custom labscript device drivers, you might only fork and clone the labscript-devices repository. Moreover, there is now an option to write and use custom labscript device drivers outside of the labscript-devices installation directory.

In this example, we will use an existing conda environment named `py38`. Skip the first line/step if continuing on from the instructions to *set up this environment*.

Quick start

Note: After the first line, the current directory is ommited from the command prompt for brevity.

```
(base) C:\Users\wkheisenberg> mkdir labscript-suite
(base) > cd labscript-suite
(base) > git clone https://github.com/wkheisenberg/labscript
(base) > git clone https://github.com/wkheisenberg/runmanager
(base) > git clone https://github.com/wkheisenberg/blacs
(base) > git clone https://github.com/wkheisenberg/lyse
```

(continues on next page)

(continued from previous page)

```
(base) > git clone https://github.com/wkheisenberg/runviewer
(base) > git clone https://github.com/wkheisenberg/labscript-devices
(base) > git clone https://github.com/wkheisenberg/labscript-utils
(base) > conda activate py38
(py38) > conda config --env --add channels labscript-suite
(py38) > conda install setuptools-conda pyqt pip desktop-app
(py38) > setuptools-conda install-requirements ^
         labscript runmanager blacs lyse runviewer labscript-devices labscript-utils
(py38) > pip install --no-build-isolation --no-deps ^
         -e labscript -e runmanager -e blacs -e lyse ^
         -e runviewer -e labscript-devices -e labscript-utils
(py38) > labscript-profile-create
(py38) > desktop-app install blacs lyse runmanager runviewer
```

Detailed instructions

- Under construction*

1.2.2 Recent changes to the *labscript suite*

Upon migrating the code base to GitHub and publishing distributions on PyPI in April–May 2020, existing users should be aware of the following recent changes.

Profile directories

The *labscript suite* profile directory, containing application configurations, logs, and user-side code, is now located by default in the current user's home directory, e.g. for a local user named `wkheisenberg` this is:

- `C:\Users\wkheisenberg\labscript-suite` on Windows.
- `~/labscript-suite` or `/home/wkheisenberg/labscript-suite` on Linux and Mac OS X.

A typical structure of the profile directory is:

```
~/labscript-suite/
├── app_saved_configs/
│   ├── default_experiment/
├── labconfig/
├── logs/
└── userlib/
    ├── analysislib/
    ├── labscriptlib/
    ├── pythonlib/
    └── user_devices/
```

This structure is created by calling the command `labscript-profile-create` in a terminal after installing `labscript-utils` (per the installation instructions).

Note: As of labscript-suite/labscript-utils#37 an editable installation can be located within the labscript-suite profile directory.

Secure communication

Interprocess communication between components of the *labscript suite* is based on the ZeroMQ (ZMQ) messaging protocol. We have supported secure interprocess communication via encrypted ZMQ messaging since February 2019 (labscript-utils 2.11.0).

As of labscript-utils 2.16.0, **encryted interprocess communication will be the default**. If you haven't already, this means you'll need to create a new shared secret (or pre-shared key) as follows:

1. Run `python -m zprocess.makesecret` from the labconfig directory.
2. Specify the path of the resulting `shared_secret` in your labconfig. For example:

```
[security]
shared_secret = %(labscript_suite)s/labconfig/zpsecret-09f6dfa0.key
```

3. Copy the same pre-shared key to all computers running the *labscript suite* that need to communicate with each other, repeating step 2 for each of them.

Treat this file like a password; it allows anyone on the same network access to *labscript suite* programs.

If you are on a trusted network and don't want to use secure communication, you may instead set:

```
[security]
allow_insecure = True
```

Notes:

- Steps 1 and 2 are executed automatically as part of the `labscript-profile-create` command. However, for multiple hosts, step 3 above must still be followed to ensure the same public-key is used by all hosts running *labscript suite* programs.
- There is an outstanding issue with the ZMQ Python bindings on Windows (zeromq/pyzmq#1148), whereby encryption is significantly slower for Python distributions other than Anaconda. Until this issue is resolved, we recommend that Windows users on an untrusted network use the Anaconda Python distribution (and install `pyzmq` using `conda install pyzmq`).

Application shortcuts

Operating-system menu shortcuts, correct taskbar behaviour, and environment activation for the Python GUI applications (blacs, lyse, runmanager, and runviewer) is now handled by a standalone Python package desktop-app (per installation instructions above). This currently supports Windows and Linux (Mac OS X support is forthcoming).

Lab configuration

The `experiment_name` item has been renamed to `apparatus_name` in the labconfig (.ini) file, to better reflect the distinciton between the infrasturcture that experiment shots are executed on. The old keyword can still be used for this item, but a warning will be issued to remind you to update your labconfig.

Source code structure (developer installation)

Existing users who move to a developer (editable) installation, please note the following structural changes to the *labscript suite* source code:

- Each package has a top-level folder containing setup.py and setup.cfg used to build a distribution from source. The functional code base now resides in a subfolder corresponding to the name of the Python module, e.g. an editable installation might contain folders:

```
<path-to-your-labscript-installation>/
├── blacs/
│   └── blacs/
├── labscript/
│   └── labscript/
├── labscript-devices/
│   └── labscript_devices/
├── labscript-utils/
│   └── labscript_utils/
├── lyse/
│   ├── lyse/
├── runmanager/
│   └── runmanager/
└── runviewer/
    └── runviewer/
```

- Package names (shared by repositories and top-level folders) are now hyphenated, e.g. labscript-devices and labscript-utils.
- Module names remain underscored, e.g. labscript_devices and labscript_utils.
- The mixing of hyphen and underscores is inelegant but conventional.
- All references to blacs are now lowercase.
- As installation no longer requires a separate package, the repository formerly named 'installer' has been renamed to 'labscript-suite', and is a metapackage for the *labscript suite* (installing it via `pip`/`conda` installs the suite).

Versioning (developer installation)

Aside from the maintenance branches described here, versions of the labscript suite packages are introspected at run-time using either the importlib.metadata library (regular installations) or setuptools_scm (developer installations). Thus any changes to an editable install will be traceable by local version numbers, e.g. editing the released version of a package with version 2.4.0 will result in 2.4.0dev1+gc28fe94, for example. This will help us diagnose issues users have with their editable installations.

1.2.3 Contributing to the *labscript suite*

We are very grateful for all the contributions users have made in the past decade to make the *labscript suite* the most widely used open-source experiment control and automation system in quantum science. These include development, suggestions, and feedback, and we look forward to this continuing on GitHub.

Issue tracking

The issue tracking on GitHub is very similar to BitBucket, with the added advantage that you can add inter-repository issue references, e.g. referring to labscript-suite/runmanager#68 in any issue or pull request will link to the corresponding issue. We have imported all issues from the BitBucket repositories into the GitHub repositories. This import is not perfect (as each comment is now posted by Phil Starkey) but the comments have been modified to contain the original author attribution. We have also updated all links to files, pull requests, issues, and commits so that they point to the equivalent GitHub location and/or the archived copy of the data (as discussed above).

Please use the issue tracker of the relevant GitHub repository for:

- Reporting **bugs** (when something doesn't work or works in a way you didn't expect);
- Suggesting **enhancements**: new features or requests;
- Issues relating to **installation**, **performance**, or **documentation**.

For advice on *how* to use the existing functionality of the *labscript suite*, please use our mailing list.

Request for developers

We would like to reaffirm our invitation for users to directly contribute toward developing the *labscript suite*. We have established a separate discussion forum on Zulip for discussing development direction and design. If you are interested in being a part of these discussions, and/or testing and merging pull requests, please reach out to us.

Pull requests

We will continue the same feature-branch workflow as before:

1. Fork one or more of the labscript suite repositories;
2. Create a branch on your fork for a new feature;
3. Make and commit changes to this branch; and
4. Make a pull request back to our repository.

These steps are broadly covered in the GitHub Hello World guide, and in detail on the NumPy development workflow.

Branching model/strategy

The move to GitHub for source control and PyPI for distribution is accompanied by a slight change in the branching strategy, to improve deployment and stability of the *labscript suite*. Whereas before all versions corresponded to single commits on the master branch; dedicated branches will be used to release and service minor versions. For example, releasing v0.1.0 would see the creation of a branch named maintenance/0.1.x, used to service all 0.1 versions. As we adhere to semantic versioning, bug-fixes would be applied in this branch, bumping the minor (final) version number each time, e.g. 0.1.1, 0.1.2, etc. No development will occur in these branches; new features are merged into master, and bug-fixes are cherry-picked from master.

You can learn more about this branching model at:

- releaseflow.org
- NumPy development workflow
- Release Flow – Azure DevOps

Learning Git

As our former development, installation, and upgrading practices involved Mercurial revision-control, some of you may not be familiar with Git. While you no longer need to use *any* revision control system to use the *labscript suite*, those of you wanting to contribute to development who aren't acquainted with Git may benefit from these resources:

- NumPy: Getting started with Git development
- GitHub Guides: Very cogent information for beginners. We recommend starting with:
  - Hello World
  - Git Handbook *Note:* You may notice references to 'GitHub Flow' in these guides (and 'Git Flow' elsewhere). These share some aspects of the Release Flow branching-workflow we use, but are distinct.
- Atliassian Git Tutorials: Despite the many references to BitBucket (ignore these); there is a wealth of excellent beginner information for using Git at the command line here; and finally
- Oh Shit, Git!?! Mistakes happen. This is a good place to start fixing them. (Censored version.)

1.2.4 BitBucket archive

In April–May 2020 the *labscript suite* code base was migrated from BitBucket to GitHub. All commit history and issues was preserved, however some repository metadata (such as pull request discussions) could not be migrated directly. As such, we have created an archived copy of everything that was on BitBucket. This includes:

- Issues (as they appear on BitBucket);
- Pull requests discussions;
- Commit comments for every labscript suite repository; and
- Every public fork (as of 1st February, 2020).

This archive can be found at bitbucket-archive.labscriptsuite.org (this page can take some time to load for the first time). Copies of every public fork of our repositories are at github.com/labscript-suite-bitbucket-archive. As this is an archive, we will not be transferring ownership of these repositories back to their original owners. However, should you wish to continue development on one of those repositories you can fork it into your own account through the GitHub web interface. Should you have uncommitted changes (or changes made after 1st February, 2020) that you wish to have archived, please contact us to discuss the best approach to including these. Please note that we are not recommending continuing development in such forks long term, due to the changes in package structure outlined above.

What to do if you had custom code in a fork on BitBucket

labscript experiment scripts and lyse analysis scripts can be copied or moved to the new labscriptlib/analysislib folders. We deem these user-side code as they are not within the codebase of the labcript suite programs, and thus do not require a developer (editable) installation.

Customisations of the labscript suite will need to be reintegrated into the new package structure, using a developer installation. For example, to include your own custom labscript devices, you should undertake the developer installation procedure for the labscript-devices repository, and copy your custom or modified device files into the labscript_devices folder alongside the existing device files. Please also consider contributing these back to the main project by pushing them to your fork and issuing a pull request.

The procedure for migrating customisations of other components will depend on how up-to-date your fork is. Please open a thread on the mailing list to discuss with us how to migrate your custom features and/or how to contribute them back to the base *labscript suite* repositories.

Migrating other repositories to GitHub

Should you have other repositories on BitBucket such as labscriptlib, analysislib, userlib, or labconfig (or any project unrelated to the *labscript suite*) we strongly suggest using the tools we developed to migrate the *labscript suite* to GitHub. These are philipstarkey/bitbucket-hg-exporter and chrisjbillington/hg-export-tool which can be used together. See the documentation of those projects for further details.

1.2.5 *labscript suite* components

The *labscript suite* is modular by design, and is comprised of:

Table 1: Python libraries

	labscript — Expressive composition of hardware-timed experiments
	labscript-devices — Plugin architecture for controlling experiment hardware
	labscript-utils — Shared modules used by the *labscript suite*

Table 2: Graphical applications

	runmanager — Graphical and remote interface to parameterized experiments
	blacs — Graphical interface to scientific instruments and experiment supervision
	lyse — Online analysis of live experiment data
	runviewer — Visualize hardware-timed experiment instructions