cyckei

Release 1.0.1

Nov 04, 2022

Contents:

CHAPTER 1

Host System Setup

Although Cyckei is developed on and for a variety of platforms, most internal usage and testing is done on Windows 10 running the latest release of Python 3. Other platforms may require more complex configuration and additional stability testing.

Cyckei relies on the PyVISA wrapper to communicate with any devices, and generally requires an additional VISA library as well as a driver for the device or adaptor which PyVISA controls. If the PyVISA python library is installed, you can use the following code in a python interpreter to list the devices which it detects. If nothing is found, proceed with the Host System Setup.

```
import pyvisa
rm = pyvisa.ResourceManager()
print(rm.list_resources())
```

Installing the necessary drivers can be difficult depending on your system. The National Instruments GPIB-USB-HS adaptors that we use require both a VISA library as well as a GPIB driver to function with PyVISA, Cyckei's core library. Installing each piece of software for different configurations is summarized below.

1.1 VISA

1.1.1 PyVISA-py

PyVISA-py is a pure python backend for PyVISA. It offers less functionality than NI-VISA, but appears to work fine with Cyckei based on limited testing. More information about PyVISA-py and installation instructions can be found in their documentation.

1.1.2 NI-VISA

NI-VISA is the VISA library developed by National Instruments it is closed source and only available for certain platforms, but offers the most functionality. NI-VISA can be downloaded at this site.

1.2 GPIB

1.2.1 NI-488.2

Like NI-VISA, NI-488.2 is National Instruments' GPIB driver. It is simple to install, but has very limited compatibility especially on Linux. Downloads for NI-488.2 can be found here.

1.2.2 Linux-GPIB

Linux-GPIB is a GPL licensed GPIB support package for Linux. In addition to the C API, it includes bindings for multiple languages including Python. Linux-GPIB must be compiled for your OS and requires some configuration, but works fine with PyVISA. To learn more about Linux-GPIB and download the source code, visit sourceforge.

CHAPTER 2

Installation

2.1 For Users and Developers

The Cyckei source code is available on GitHub at our public repository , and can be cloned locally to run the latest version.

```
git clone https://github.com/cyclikal/cyckei.git
cd cyckei
```

Cyckei requires Python 3 in addition to some packages which can be installed via pip and the included requirements file. Consult `setup.py` for a complete list of requirements.

Python can be run directly from source using the `cyckei.py` script in the root of the repository.

```
python cyckei.py
```

For more information about editing and contributing to Cyckei see *Contributing*.

CHAPTER 3

Using Cyckei

3.1 First Launch

Cyckei can be launched for the first time from the root folder using:

```
python cykei.py
```

Upon first launch, Cyckei will create a `cyckei` directory in the user's home folder to hold scripts, test results, logs, and configuration. Cyckei will also create a server_data text file that facilitates the clients memory of the server's activities.

Before running tests, Cyckei must be configured to properly interface with any devices. Each channel should be setup in the `config.json` file with the correct GPIB address and any other relevant information. A default configuration is automatically generated, and instructions on further configuration can be found in the *Editing Configuration* section.

3.2 Client, Server, and Explorer

Cyckei comes with three sibling applications: A server, a client, and the explorer. The server and client act in tandem, while the explorer is independent. Server performs the work of sending commands to cycle cells, while the client provides the user an interface to interact with the server. The explorer is used for viewing completed tests and creating new scripts to be run by the server.

The server should be launched before a client from the root directory with

```
python cyckei.py server
```

If a client does not have a server to connect to, it will be essentially non functional. After the server is launched the client can be launched from the root directory with

```
python cyckei.py client
```

Finally, the explorer can be launched from the root directory with

```
python cyckei.py explorer
```

On Windows a bash file can be set up as a shortcut to run each command sequence.

3.3 Starting a cycle

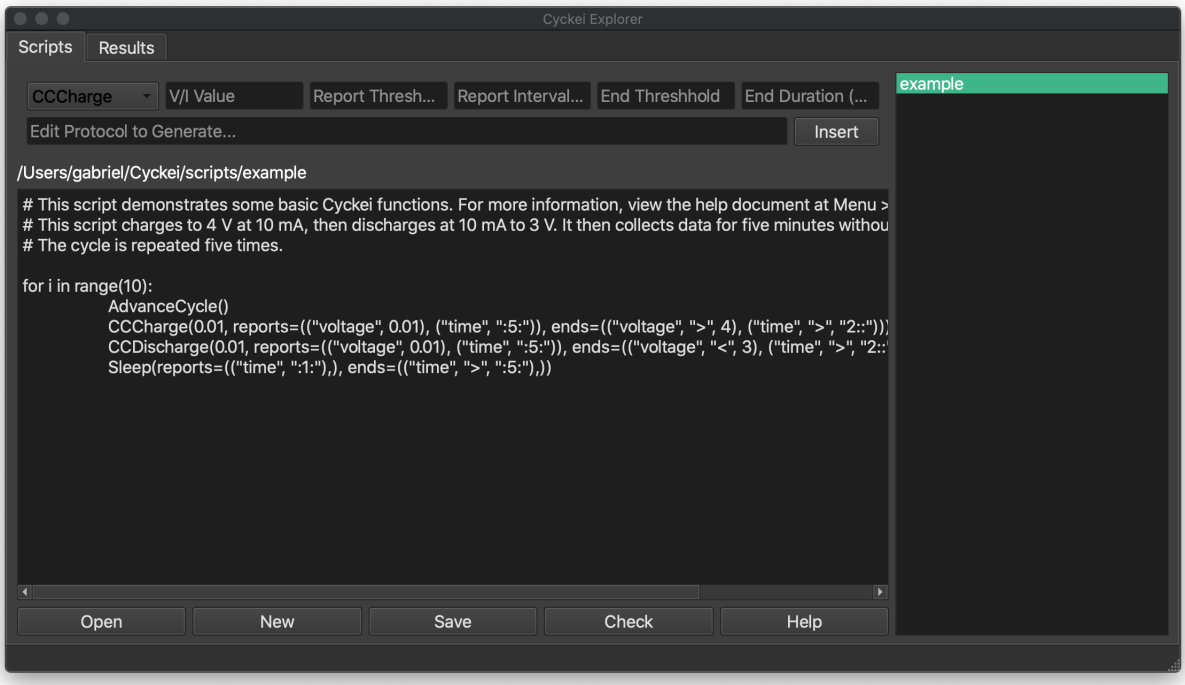
Various attributes of the cycle may be set in before starting a cycle:

Option	Type	Description
Script	file	Script with desired protocol. Gives option to select any local file.
Log file	text	Path to output file. Placed in the specified logs folder.
Cell ID	text	Identification for cell. Recorded to output file.
Comment	text	Requester's comment for cycle. Recorded to output file.

The available buttons can be used to Start, Stop, Pause, or Resume the protocol.

3.4 Creating Scripts

Scripts can be created in the user's preferred editor or in the separate explorer application. This editor will automatically load the default included scripts, but can be used to open and edit any local files.



The explorer includes a protocol generator above the editor to streamline script creation. This can be used to specify attributes, and insert generated lines of code into the script.

Scripts are written in regular python code, and can contain for loops and other statements to control cycle flow. There are seven built in protocols to control the cycler. Most of these protocols take some or all of the following parameters:

Parameter	Description	Format
Value	Set a certain voltage or current to run at.	float
Reports	Set intervals of time and/or change in voltage or current to record at.	reports=((“current”, *float*), (“time”, “*int*:*int*:*int*”))
Ends	Set threshold of time and/or change in voltage or current to end current protocol.	ends=((“current”, “<”, *float*), (“time”, “>”, “*int*:*int*:*int*”))

The following protocols are available:

Protocol	Description	Parameters	Example
AdvanceCycle	Start recording under next cycle in output file.	None	AdvanceCycle()
CCCharge	Charge at a set current.	Value, Reports, Ends	CCCharge(0.1, reports=((“voltage”, 0.01), (“time”, “:5:”)), ends=((“voltage”, “>”, 4.2), (“time”, “>”, “4::”)))
CCDischarge	Discharge at a set current.	Value, Reports, Ends	CCDischarge(0.1, reports=((“voltage”, 0.01), (“time”, “:5:”)), ends=((“voltage”, “<”, 3.0), (“time”, “>”, “4::”)))
CVCharge	Charge at a set voltage.	Value, Reports, Ends	CVCharge(4.2, reports=((“current”, 0.01), (“time”, “:5:”)), ends=((“current”, “<”, 0.005), (“time”, “>”, “24::”)))
CVDischarge	Discharge at a set voltage.	Value, Reports, Ends	CVDischarge(4.2, reports=((“current”, 0.01), (“time”, “:5:”)), ends=((“current”, “<”, 0.005), (“time”, “>”, “24::”)))
Rest	Record at a set interval.	Reports, Ends	Rest(reports=((“time”, “::1”),), ends=((“time”, “>”, “::15”),))
Sleep	Record at a set interval and turn channel off in between.	Reports, Ends	Sleep(reports=((“time”, “:1:0”),), ends=((“time”, “>”, “::15”),))

An example script is shown below. There is also a simple script saved in the scripts folder which is available whenever the client is started.

```
for i in range(3):
  AdvanceCycle()
  CCCharge(0.1, reports=(("voltage", 0.01), ("time", ":5:")), ends=(("voltage", ">",␣
↪4.2), ("time", ">", "4::")))
  CCDischarge(0.1, reports=(("voltage", 0.01), ("time", ":5:")), ends=(("voltage", "<
↪", 3.0), ("time", ">", "4::")))
  Rest(reports=(("time", "::1"),), ends=(("time", ">", "::15"),))
```

It is important to note that variables cannot be assigned in the standard pythonic way

```
C = 0.1
```

However, for loops can be used to capture values as variables as shown in this next example where C is caputred as 0.1 A and substituted in for C in the CCCharge and CCDischarge protocols.

The example also shows the nesting of loops. In this case a total of 500 cycles would be completed, where C/4 cycling is done with a C/20 cycle every 50 cycles.

```
for C in [0.1]:
    for i in range(10):
        AdvanceCycle()
        CCCharge(C/20, reports=(("voltage", 0.005), ("time", ":5:")), ends=(("voltage
↪", ">=", 4.2), ("time", ">", "30::")))
        CCDischarge(C/20, reports=(("voltage", 0.005), ("time", ":5:")), ends=((
↪"voltage", "<", 3), ("time", ">", "30::")))
        for j in range(49):
            AdvanceCycle()
            CCCharge(C/4, reports=(("voltage", 0.005), ("time", ":1:")), ends=((
↪"voltage", ">=", 4.2), ("time", ">", "6::")))
            CCDischarge(C/4, reports=(("voltage", 0.005), ("time", ":1:")), ends=((
↪"voltage", "<", 3), ("time", ">", "6::")))
```

Access to the python interpreter allows powerful options. The next example shows testing of rate capability in a convenient loop. Three cycles are completed at discharge rates of C/20, C/10, C/5, C/2, and C with the charge remaining C/20 in all cases.

```
for C in [0.1]:
    for X in [20,10,5,2,1]:
        for i in range(3):
            AdvanceCycle()
            CCCharge(C/20, reports=(("voltage", 0.005), ("time", ":5:")), ends=((
↪"voltage", ">=", 4.2), ("time", ">", "30::")))
            CCDischarge(C/X, reports=(("voltage", 0.005), ("time", ":5:")), ends=((
↪"voltage", "<", 3), ("time", ">", "30::")))
```

Scripts are automatically checked when they are sent to the server. They can also be manually checked by clicking the "Check" button below the editor. Checking a script ensures that (1) the script only contains legal arguments and (2) can be loaded by the server without immediate errors. Checking your scripts is a good practice to mitigate possible formatting issues and errors. However, care should still be taken while writing scripts as they are executed as any other python code within the application.

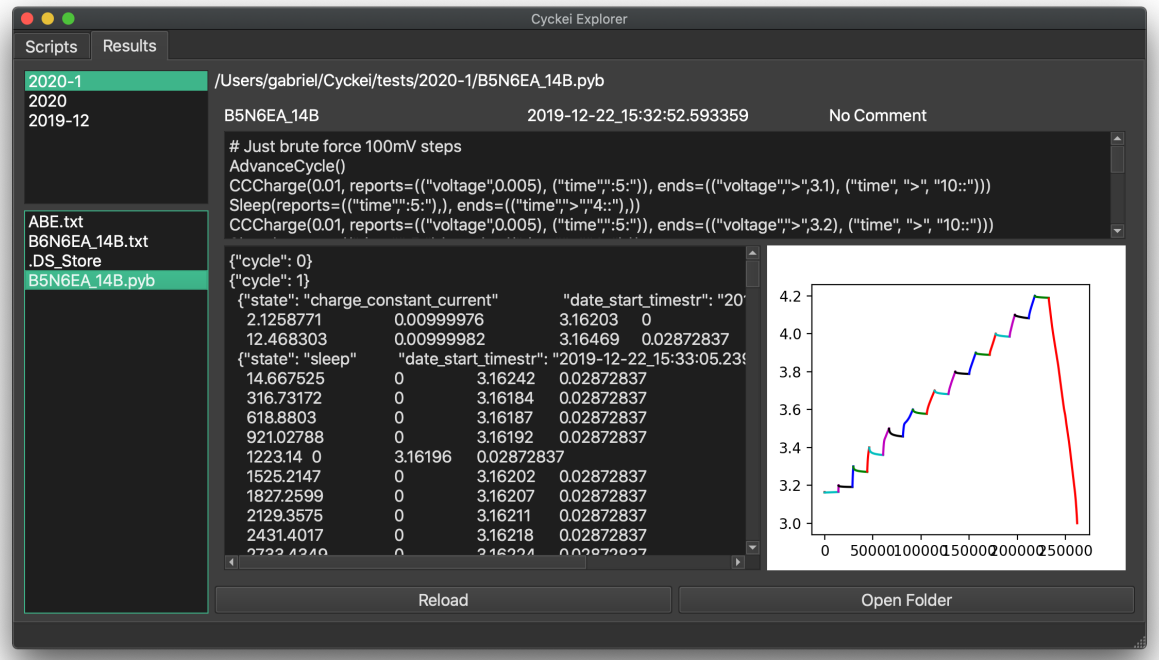
3.5 Using Plugins

Data plugins are available to supplement current and voltage data measurements. The plugin scheme is designed to be flexible in order to support any device with the use of custom configuration. A random plugin is included by default with the Cyckei distribution. Other plugins can be written by developing a similar DataController object and including it in the `plugins` folder of the Cyckei recording directory. Below is an example plugin for reference.

```
import logging
from random import randint

logger = logging.getLogger('cyckei')

class DataController(object):
    def __init__(self):
        self.name = "random"
        logger.info("Initializing Random Recorder plugin")

    def read(self):
        logger.debug("Generating random integer...")
        return randint(1, 101)
```

3.6 Viewing Results

Results are created to document measurements from each cell throughout it's cycle. They also have details about the cell and the cycle that was run on it. Result files are saved to the `tests` folder specified in the configuration under the specified name. To view a result file from the client application, just open the explorer application. All result files are automatically loaded on startup in the explorer application, and new or updated ones can be viewed after clicking reload. Although you can copy the contents of a result file to an excel spreadsheet, result files *should not* be opened with excel or another application directly. Doing this can cause the file to become locked and prevent Cyckei from editing it.



3.7 Viewing Logs

Log text files are stored in the `logs` folder in Cyckei. These logs capture information about the exectuion of their respective program. For example: server or client logs. In these files Errors, Warnings, and different steps in the exectution of the programs are stored.

3.8 Editing Configuration

Editing the configuration file is crucial for the client to function properly. Any custom configuration files should be written in JSON and should mirror the default config.json in the program's root directory. Each section is described in more detail below:

- **channels** - A list of channels currently connected to the computer.
  - *channel (string)* - Channel number for identification within the application.

  - *gpib_address (int)* - Hardware address of GPIB interface can be found with a NI VISA application or wth the code in *Plugin Overview*.
  - *keithley_model (string)* - Model number of keithley being used.
  - *keithley_channel (string)* - Particular channel on said keithley (a or b).

- **zmq** - A dictionary of properties that control how the client and server communicate.

  zmq is now stored in variables.ini in the cyckei assets file

  - *port (int)* - Port to communicate over.
  - *client-address (string)* - Address for the client to connect to. Usually localhost.
  - *server-address (string)* - Address for the server to listen on. Usually all.
  - *timeout (int)* - Number of seconds to wait for server response. 10 seconds seems to work well for most configurations.

- **data-plugins** - A list of data plugins to load and execute alongside normal data collection. Plugins should be placed in the `plugins` directory of the Cyckei recording folder.
- **device** - The identifier for which device to load. Currently, `keithley2602` is the only acceptable model.
- **verbosity** - The amount of information to be saved to log files. Generally should be set to 20, but the following levels can also be used. Lower values print more information for debugging purposes.
  - *Critical* - 50
  - *Error* - 40
  - *Warning* - 30
  - *Info* - 20
  - *Debug* - 10
  - *Notset* - 0

Here is an example configuration file for a simple setup running on port 5556 with one Keithley with address 5:

```
{
    "channel_readme": "List of keithley channels to connect.",
    "channels": [
      {
        "channel": 1,
        "gpib_address": 9,
        "keithley_channel": "a",
        "model": "2602B"
      },
      {
        "channel": 2,
        "gpib_address": 9,
        "keithley_channel": "b",
        "model": "2602B"
      },
      {
        "channel": 3,
        "gpib_address": 5,
        "keithley_channel": "a",
        "model": "2602B"
      },
      {
```

(continues on next page)

(continued from previous page)

```
        "channel": 4,
        "gpib_address": 5,
        "keithley_channel": "b",
        "model": "2602B"
      }
    ],
    "zmq":{
        "port": 5556,
        "client-address":"tcp://localhost",
        "server-address":"tcp://*",
        "timeout": 10
    },
    "plugins_readme": "List of plugins to connect, each declaring sources.",
    "plugins": [
      {
        "name": "randomizer",
        "module": "randomizer",
        "enabled": false,
        "sources": [
          {
            "port": null,
            "meta": [1, 10]
          },
          {
            "port": null,
            "meta": [11, 20]
          }
        ]
      }
    ],
    "verbosity": 30
}
```

CHAPTER 4

Plugins

4.1 Plugin Overview

Although the base version fo Cyckei is designed to control and capture data from Keithley SourceMeters, the functionality can be extended with plugins. Plugins aim to support simultaneous data collection with additional instruments that can interface with a computer. This requires that the data capturing eventually be queried and returned by python code, but the method by which this is done is very flexible.

4.2 Installation & Configuration

Plugins are distributed independently of Cyckei as python packages. These packages are generally able to be executed independently for testing purposes, but are designed to be loaded by Cyckei for data capture.

Generally, installation involves cloning the github repository, changing to the directory and installing the python package:

To be loaded by cyckei, an entry must also be added to the `plugins` section of Cyckei's `config.json`. Plugins should provide an example configuration and instructions on how to adjust it. The configuration for the [randomizer] plugin is included for reference.

```
{
  "name": "randomizer",
  "enabled": true,
  "sources": [
    {
      "port": null,
      "meta": [1, 10]
    },
    {
      "port": null,
      "meta": [11, 20]
    }
```

(continues on next page)

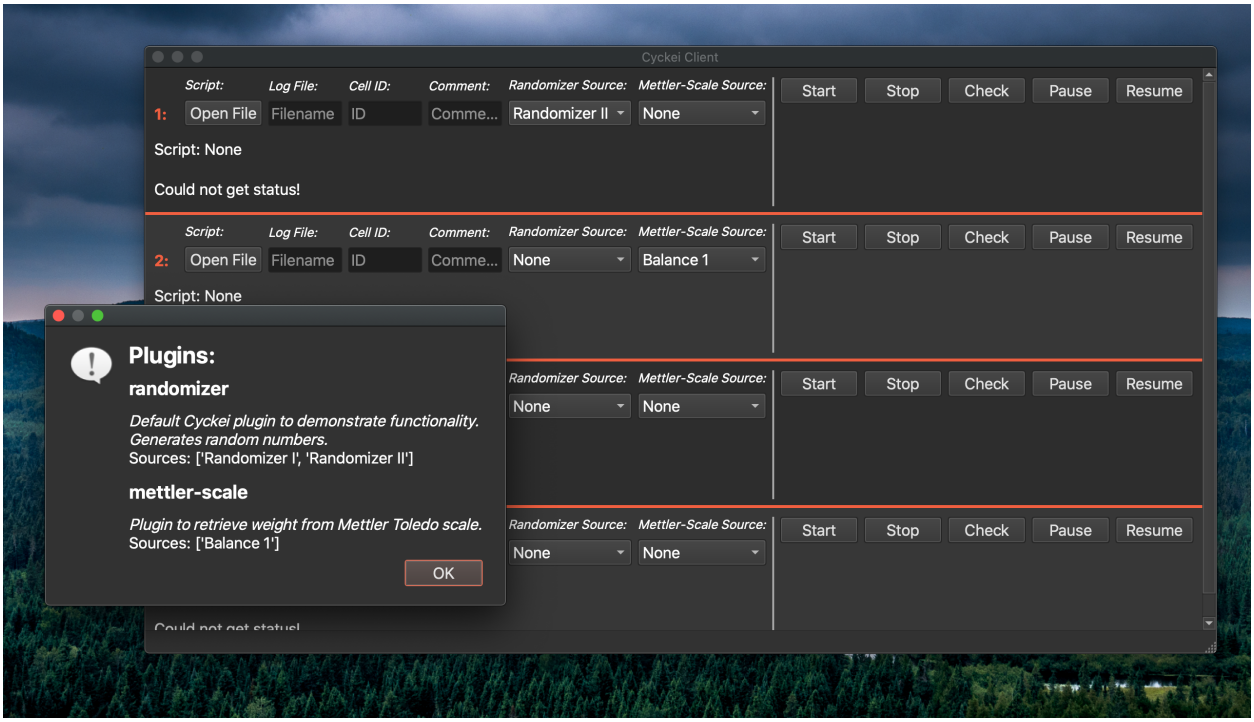
(continued from previous page)

```
    ]
}
```

The configuration includes a number of reference values such as a name, whether the plugin should be enabled. It also has a list of sources that can be assigned to different channels. The `mettler-ag204` plugin, for example, has the ability to interface with multiple Mettler Toldedo AG-204 scales, and declares them as several sources. The Cyckei interface then has the ability to assign different scales to individual channels for data capture. The exact parts of each source entry may depend on the individual plugin, but a port number and meta information are pretty standard. Port numbers and other information should be changed as necessary for your setup.

4.3 Running

Once configured, the different data sources exposed by plugins will be visible in the Cyckei Client.



Once available, it is as simple as selecting the source in the corresponding dropdown to assign a device to each channel. Once assigned, data from the device will be merged into the output file for that channel.

4.4 Available Plugins

The following plugins are currently known to be available. Submit a pull request to add or update entries for custom plugins.

An example of using interpreted text

Name	Purpose	Source	Version
Randomizer	Example, produces random numbers.	GitHub	0.1 Stable
Mettler AG-204	Weight from Mettler Toledo AG-104.	GitHub	0.1 Stable
Pico TC-08	Temperature from Pico TC-08 Thermocouples.	GitHub	0.1 Stable
Novus-n1050 PID	Reads data from connected Novus-n1050 PID.	GitHub	0.1 Stable

4.5 Custom Plugins

Custom plugins are simple to create, especially if there is an established method of reading device data into python already. It is recommended that you follow the scheme of the [Randomizer Plugin](https://github.com/cyclikal/cyp-randomizer).

The main component of any plugin is the `PluginController` class. This class is a child of Cyckei's BaseController class which provides a number of helper functions including the essential `read()` method. The `cyp-randomizer` package includes in-line documentation to demonstrate the changes that need to be made to create a plugin for a new device. Generally most setup should be performed in the `load_sources()` method, and any steps to capture data should occur in the `read()` method. It is good practice to create some basic documentation to accompany a custom plugin, particularly if additional drivers need to be installed. Without sufficient documentation it is unlikely that plugins will be officially supported.

Another good example is the `mettlerscale` plugin, which gathers data from a Mettler-Toledo balance. In addition to having a `read()` function, this plugin utilizes a `MettlerLogger` object to interact with each individual scale on a different port.

CHAPTER 5

Contributing

5.1 Developing Cyckei

Cyckei is an open source project created with the intention of allowing users to customize it to fit their equipment and setup. We encourage users with some programming knowledge to contribute changes that may be helpful to the community.

5.2 Workflow

GitHub hosts and manages version control for the Cyckei source code. The Cyckei project is kept under Cyclikal LLC's repository. To submit patches to the codebase, fork the project to a personal GitHub account and make any desired changes. After completing a bug fix or new feature, open a new merge request into Cyclikal's repository along with thorough documentation. After review, the patch may integrated into Cyckei's development branch.

CHAPTER 6

Changes & Features

6.1 0.0 Yin - 11/14/2018

Original version of Cyckei.

6.1.1 Notable Changes

- Create complex UI to handle all software functions
- Implement cycling protocols such as CCCharge and Sleep

6.2 0.1 Vayu - 07/2/2019

Intended to significantly improve the performance and responsiveness of the application by improving the execution pattern and introducing threading to the Qt interface. Also overhauls the UI and brings many components up to date.

6.2.1 Notable Changes

- Run client communication functions as synchronous worker
- Switch GUI from PyQt5 to PySide2
- Improve layout and scaling of UI elements

6.2.2 Development Releases

- 0.1.dev1, 05/26/2019 – Initial Cyclikal commits
- 0.1.dev2, 05/30/2019 – Adjust layout and switch to PySide2
- 0.1.dev3, 05/31/2019 – Create threaded workers for each action

- 0.1.dev4, 06/02/2019 – All primary buttons execute as separate thread
- 0.1.dev5, 06/03/2019 – Message Boxes and status updates are sent through signal/slot pattern
- 0.1.dev6, 06/12/2019 – Overhaul visual appearance for simplicity
- 0.1.dev7, 06/12/2019 – Separate client and server packages for proper file access during distribution
- 0.1.dev8, 06/13/2019 – Fix over-threading and application exit
- 0.1.dev9, 06/27/2019 – Move server to applet and improve OS integration
- 0.1.dev10, 06/27/2019 – A bunch of script tab fixes and separated status and feedback on the channel tab

6.2.3 Release Candidates

- 0.1rc1, 06/28/2019 – Initial Release Candidate
- 0.1rc2, 06/29/2019 – Fixed some bugs, enable MenuBar on Windows, and added exception logging

6.3 0.2 Alviss - 8/01/2019

Smaller update focused on simplifying the code to aid in further development. This includes unifying as many commonly used functions as possible and adding code documentation. Also adds single file executables because they're fun.

6.3.1 Notable Changes

- Unify common functions and generally refactor codebase
- Support distribution of compiled executables
- Improve documentation
- Small UI adjustments including dark mode
- Rewrite "Read" and "StatusUpdate" functions for better performance and functionality

6.3.2 Development Releases

- 0.2.dev1, 7/15/2019 – Improve Documentation
- 0.2.dev2, 7/17/2019 – Switch to PyInstaller build system
- 0.2.dev3, 7/20/2019 – Simplify client codebase, unify common functions, improve UI
- 0.2.dev4, 7/21/2019 – Introduce Sphinx and add contribution documentation
- 0.2.dev5, 7/24/2019 – Small adjustments to prepare release candidates

6.3.3 Release Candidates

- 0.2rc1, 7/24/2019 – Fix some small bugs
- 0.2rc2, 7/24/2019 – Fix bugs, reduce server calls, and document issues
- 0.2rc3, 7/30/2019 – Improve status updates and "Read Cell" function

- 0.2rc4, 7/31/2019 – Fix file naming while reading cell, unify versioning
- 0.2rc5, 8/01/2019 – Report pre-logging runtime errors

6.4 0.2.1

Hotfix updates to Alviss. Includes fixes to delay and improper current measurement and a basic test suite.

6.4.1 Notable Changes

- Fixes to measurement and report timing
- Basic test suite

6.4.2 Development Releases

- 0.2.1.dev1, 8/19/2019 – Fix issues with delay and improper current measurement

6.5 0.3 Tenzin

Rebuilding existing interfaces after fixing an OS-level threading error. Adds Cyckei Explorer for editing scripts and viewing recent log files. Now, again, uses PyPI release system as opposed to freezing.

6.5.1 Development Releases

- 0.3.dev1, 1/16/2020 – Implements Cyckei Explorer and rebuilds distribution system
- 0.3.dev2, 1/31/2020 – Rebuild check, pause, & resume functionality
- 0.3.dev3, 3/28/2020 – Add plugin scheme to support additional data collection
- 0.3.dev4, 0/00/0000 – Highlight scripts in explorer & Fix Flickering

6.5.2 Release Candidates

- 0.3rc1, 0/00/0000 – TBD

6.6 0.4 Skyler

Adds plugin scheme.

- 0.4.dev1, 6/29/2020 – Implement Plugin System
- 0.4.dev2, 7/22/2020 – Fix Metadata Values
- 0.4.rc1, 8/12/2020 – Package-based Plugin System
- 0.4.rc2 8/25/2020 – Update documentation for Cyckei and Plugins

6.7 0.5 Themis

Stability Update

Added testing schemes for the server and client of Cyckei. Bug fixes catching user input and preventing crashes. Logging is now done separately for separate aspects of Cyckei. Added usability to Cyckei client: saving entered info, channel locking when a script is run, visual indications of runnings scripts, and more. Fully documented server and client.

6.8 Possible Features

- **Client Interface**
  - Better batch management
  - Multi-folder script storage
  - Script highlighting
- **Server Software**
  - "Plug-in" style core (lua) script management for different devices
  - Implement Cython and threading for improved performance with massive cycles
- **Hardware Support**
  - Complete Support for Linux
  - Simplify VISA and driver installation for end user
- **Miscellaneous**
  - Automated release delivery

CHAPTER 7

Codebase

7.1 Main

This file is for execution as an installed package via 'cyckei'.

class cyckei.cyckei.**ColorFormatter**(*fmt=None*, *datefmt=None*, *style='%'*)
Extends logging.Formatter. Formatter to add colors and count warning / errors.

Set as the formatter for loggers when they are initalized in start_logging().

format(*record*)
Called by the logger this object is attached to to format records.
Parameters **record** (*logging.LogRecord*) – The record to be formatted.
Returns The Formatter to be used by loggers.
Return type str

cyckei.cyckei.**file_structure**(*path*, *overwrite*)
Checks for existing folder structure and sets up if missing

Parameters **config** – Primary configuration dictionary

cyckei.cyckei.**handle_exception**(*exc_type*, *exc_value*, *exc_traceback*)
Exception Handler (referenced in start_logging)

cyckei.cyckei.**load_plugins**(*config*)
Takes the plugins listed in the config dict and attempts to import and instantiate them.

Parameters **config** (*dict*) – Primary configuration dictionary.

Returns

The first value is a list of PluginControllers extending the BaseController object. The second value is a dict v
a value of the of the specific plugin instance's name.

Return type (list, dict)

cyckei.cyckei.**main**(*args=None*)
The entry point for the application that controls the execution of different program branches.

Parses command-line arguments for component and directory. Checks for and, if necessary, creates file structure at given directory. Compiles configuration from config and variable files. Starts logging to both console and file based on argument input. Launches requested cyckei component (server, client, or explorer).

cyckei.cyckei.**make_config**(*args*, *logger*)
Loads configuration and variables from respective files. Merges them and adds command line arguments for universal access.

Parameters

- **args** – All processed command line arguments.
- **logger** (*Logger*) – The logger to log config creation to.

Returns Completed 'config' dictionary.

Return type dict

cyckei.cyckei.**parse_args**()
Creates and parses command line arguments

Returns ArgumentParser with filled arguments.

cyckei.cyckei.**start_logging**(*config*, *logger*)
Creates handlers and starts logging.

Logs to both file (f_handler) and console (c_console).

Parameters

- **config** (*dict*) – Primary configuration dictionary.
- **logger** (*Logger*) – The logger to initialize.

7.2 Client

Widget that controls a single channnel. Listed in the channel tab of the main window.

class cyckei.client.channel_tab.**ChannelTab**(*config*, *resource*, *parent*, *plugin_info*, *channel_info*)
Object that creates a window to interact with cycler channels from the server.

config
Holds Cyckei launch settings.
Type dict

resource
A dict holding the Threadpool object for threads to be pulled from.
Type dict

channels
A list of ChannelWidget objects.
Type list

timer
A timer for the status of cells.
Type QTimer

__init__(*config*, *resource*, *parent*, *plugin_info*, *channel_info*)
Inits ChannelTab with channels, config, resource, and timer. Creates each channel widget and place in QVBoxlayout.
Parameters
- **config** (*dict*) – Holds Cyckei launch settings.
- **resource** (*dict*) – A dict holding the Threadpool object for threads to be pulled from.
- **parent** (MainWindow) – The MainWindow object that created this ChannelTab.
- **plugin_info** (*list*) – A list of dicts holding info about installed plugins.
- **channel_info** (*dict*) – A dict of nested dicts holding info about each connected channel.

- | –

alternate_colors()
Sets the channels to alternate between light and dark.

paintEvent(*event*)
Redraws the window with the current visual settings. Overrides the defaul QT paintEvent.

update_status()
Updates the status section of a channel.

class cyckei.client.channel_tab.**ChannelWidget**(*channel*, *config*, *resource*, *plugin_info*, *cur_channel_info*)
Object that controls and stores information for a given channel.

attributes
Holds info about the ChannelWidget: Channel info, cell info, script info, etc.
Type dict

config
Holds Cyckei launch settings.
Type dict

default_color
The default background color of the channel.
Type str

divider
Divides the channel widget vertically between info and controls.
Type QWidget

feedback
A label under the controls that gives info when a control is pressed.
Type QLabel

json
Holds the default attribtues of a ChannelWidget. Taken from an outside file.
Type dict

script_label
A gui label that indicates if there is a selected script.
Type QLabel

settings
A list of gui elements to be added to the window, set in the set_settings function.
Type list

state
The step in the protocol performed on a cell.
Type str

state_changed
Indicates whether the channel state has changed.
Type bool

status
A gui label that indicates a cell's status.
Type QLabel

threadpool
Holds the Threadpool object from resource for threads to be pulled from.
Type dict

__init__(*channel*, *config*, *resource*, *plugin_info*, *cur_channel_info*)
Inits ChannelWidget with attributes, config, divider, feedback, json, script_label, status, and threadpool.
Parameters
- **channel** (*int*) – Id number for the channel corresponding with this Widget.
- **config** (*dict*) – Holds Cyckei launch settings.
- **resource** (*dict*) – A dict holding the Threadpool object for threads to be pulled from.
- **plugin_info** (*list*) – A list of dicts holding info about installed plugins.
- **cur_channel_info** (*dict*) – A dict holding info about the corresponding channel for this Widget.

button(*text*)
Controls what happens when a button on the control panel is pressed.
Creates workers and uses the threadpool to run cycler functions.
Parameters **text** (*str*) – Button text that determines which function to do.

get_controls()
Creates a set of buttons in an element that control the cycler.
Returns A list of gui buttons that control the cycler.
Return type list

get_settings(*cur_channel_info*, *plugin_info*)
Creates all UI settings and adds them to settings list.

Parameters
- **cur_channel_info** (*dict*) – A dict holding info about the corresponding channel for this Widget.
- **plugin_info** (*list*) – A list of dicts holding info about installed plugins.

Returns a list of gui elements to be added to the window.
Return type list

lock_settings()
Sets the status of each QObject in settings to be uninteractable

paintEvent(*event*)
Redraws the window with the current visual settings. Overrides the defaul QT paintEvent.

set(*key*, *text*)
Sets the attributes dict using the corresponding key and text.

Used to set ChannelWidget's script to the one selected in dropdown.

Parameters
- **key** (*str*) – The key in the attributes dict in the ChannelWidget to have its value changed.
- **text** (*str*) – The new value for the corresponding key in the attributes dict.

set_bg_color()
Checks whether the background needs to be changed and acts accordingly

set_plugin(*key*, *text*)
Sets object's plugin to one selected in dropdown

Parameters
- **key** (*str*) – The key in the "plugins" section of the attributes dict in the ChannelWidget to have its value changed.
- **text** (*str*) – The new value for the corresponding key in the "plugins" section of the attributes dict.

set_script(*button_text*, *filename=None*)
Sets the protocol for a channel to run

By default opens a finder window to select a file If a filepath is already provided then the finder window is skipped.

Parameters
- **button_text** (*str*) – The text of the button being pressed.
- **filename** (*str, optional*) – The name of the script file at the end of the stored script path. Defaults to None.

set_state(*state=None*)
Changes the state of the channel and marks if the state has been changed or not.

Parameters **state** (*str, optional*) – The step the channel protocol is on. Defaults to None.

unlock_settings()
Sets the status of each QObject in settings to be interactable

Main window for the cyckei client.

class cyckei.client.client.**MainWindow**(*config*)
An object for generating the main client window and holding information about it.

config
Holds Cyckei launch settings.

Type dict

channel_info
Holds info on channels available on the server.

Type dict

channels
A list of all of the ChannelWidgets in channelView

Type list

channelView
Wrapper object that holds all of the ChannelWidgets.

Type *ChannelTab*

status_bar
Default status bar for the QWindow.
Type QStatusBar

threadpool
Threadpool of workers for communicating with the server
Type QThreadPool

__init__(*config*)
Inits Mainwindow with config, channel_info, channels, channelView, status_bar, and threadpool.

Args: config (dict): Holds Cyckei launch settings. Is copied to MainWindow's version of config. |

closeEvent(*event*)
Overridden method from QMainWindow for closing the application.
Parameters **event** (*QCloseEvent*) – An event carrying flags for closing the Q application.

create_menu()
Sets up the menu bar at the top of the Main Window.

ping_server()
Checks for an active server and returns a result message in a new window.

plugin_dialog()
Compiles and formats info for installed plugins on the server for display.

cyckei.client.client.**main**(*config*)
Begins execution of Cyckei.

Parameters **config** (*dict*) – Holds Cyckei launch settings.

Returns Result of app.exec_(), Qt's main event loop.

class cyckei.client.socket.**Socket**(*config*)

Object that handles connection, communication, and control of server from the client over ZMQ.

config

Holds Cyckei launch settings.

Type dict

socket

The underlying scoket that acts as the communication between client and server.

Type zmq.socket

__init__(*config*)

Inits Socket with config and socket.

Parameters **config** (*dict*) – Holds Cyckei launch settings.

info_all_channels()

Sends a JSON request for information on all channel to server.

Returns A dict of nested dicts containing info about all connected channels.

Return type dict

info_channel(*channel*)

Sends a JSON request for information on a channel to server.

Parameters **channel** (*int*) – The id number of the channel to request info about.

Returns Information about the requested channel.

Return type dict

info_plugins()

Sends a JSON request for information on plugins to server.

Returns A list from the server of the plugins that are installed.

Return type list

info_server_file()
Sends a JSON request for the server file kept by the server.
Returns A dict of nested dicts containing info about all connected channels from the server file.
Return type dict

ping()
Sends a JSON "ping" to the server to check status
Returns A JSON response from the server with the port the server is on.
Return type dict

send(*to_send*)
Sends JSON packet from client to server over zmq socket.
Parameters **to_send** (*dict*) – JSON in the form of a python dict to be sent to server.
Returns A JSON response from the server in the form of a python dict.
Return type response (dict)

send_file(*file*)
Sends a JSON packet from the client to the server over a zmq socket, loaded from a file.
Parameters **file** (*str*) – File path of the JSON file to be loaded and sent to the server.
Returns A JSON response from the server in the form of a python dict.
Return type dict

Methods and object to handle scripts

class cyckei.client.scripts.**Script**(*title*, *path*)
Object for storing and manipulating strings that act as scripts

content
The text that acts as the script in the file of the script.
Type str

path
The filepath of the file of the script.
Type str

title
The filename of the script being held.
Type str

__init__(*title*, *path*)
Inits Script with content, path, and title.

Args: title (str): The filename of the script being held. path (str): The filepath of the file of the script. |

save()
Saves script content to file using the script's path and title.

update_status()
Updates the script's title with '*' if the script's contents has been edited

class cyckei.client.workers.**Check**(*config*, *protocol*)
Object used for testing whether a certain protocol can be run.

protocol
The protocol being checked for legality.
Type str

config
Holds Cyckei launch settings.
Type dict

signals
Used for gui signals. Shows the server's response.
Type *Signals*

__init__(*config*, *protocol*)
Inits Check with config, protocol, and signals.
Parameters
- **config** (*dict*) – Holds Cyckei launch settings.
- **protocol** (*str*) – The protocol being checked for legality.

legal_test(*protocol*)
Checks if script only contains valid commands.
Parameters **protocol** (*str*) – The protocol being checked for legality.
Returns True if protocol is legal, False otherwise. str: The message that goes with the legality test results.
Return type bool

prepare_json(*protocol*)
Packages protocol in json dict to send to server.
Parameters **protocol** (*str*) – The protocol being sent to the server.
Returns The protocol packaged with an indication that this is a test for the server.
Return type dict

run()
Runs the tests for checking the script.
Returns True if protocol is legal and loaded, False otherwise. str: The message that goes with the legality/load test results.
Return type bool

run_test(*protocol*)
Checks if server can load script successfully.
Parameters **protocol** (*str*) – The protocol being checked for server loading.
Returns True if protocol is loaded, False otherwise. str: The message that goes with the load test results.
Return type bool

class cyckei.client.workers.**Control**(*config*, *channel*, *command*, *script=None*, *temp=False*)
Object for storing a script and sendng it to the server.

channel
Channel object that stores info about itself.
Type *ChannelWidget*

command
Used by Control in determining which actions to take.

Type str

config
Holds Cyckei launch settings.
Type dict

script
The script for the server to execute. Passed in or taken from the channel.
Type str

signals
Used for gui signals. Shows the server's response.
Type *Signals*

temp
Indicates whether recording should be done in temporary files (true) or not (false)
Type bool

__init__(*config*, *channel*, *command*, *script=None*, *temp=False*)
Inits Control with channel, command, config, script, signals, and temp.
Parameters
- **channel** (ChannelWidget) – Channel object that stores info about itself.
- **command** (*str*) – Used by Control in determining which actions to take.
- **config** (*dict*) – Holds Cyckei launch settings.
- **script** (*str, optional*) – The script for the server to execute. Defaults to None.
- **temp** (*bool, optional*) – Indicates whether recording should be done in temporary files (true) or not (false). Defaults to False.

run()
Calls for a viability check on the loaded script and then sends it to the server.

class cyckei.client.workers.**Ping**(*config*)
Object used to check for an active server.

config
Holds Cyckei launch settings.
Type dict

signals
Used for gui signals. Shows the server's response.
Type *Signals*

__init__(*config*)
Inits Ping with signals and config.
Parameters **config** (*dict*) – Holds Cyckei launch settings.

run()
Sends a ping to the Socket to check status. Emits the response.

class cyckei.client.workers.**Read**(*config*, *channel*)
Object used in reading cell information from the server.

channel
Channel object that stores info about itself.
Type *ChannelWidget*

config
Holds Cyckei launch settings.
Type dict

signals
Used for gui signals. Shows the server's response.
Type *Signals*

__init__(*config*, *channel*)
Inits Read with channel, config, and signals.
Parameters
- **channel** (ChannelWidget) – Channel object that stores info about itself.
- **config** (*dict*) – Holds Cyckei launch settings.

run()
Tell channel to Rest() long enough to get voltage reading on cell.

class cyckei.client.workers.**Signals**
Object used by other objects to alert the user to changes, statuses, etc.

class cyckei.client.workers.**UpdateStatus**(*channels*, *config*)
Updates the status below the controls, shown after contacting server

channels
A list of all of the ChannelWidget objects to be updated.
Type list

config
Holds Cyckei launch settings.
Type dict

__init__(*channels*, *config*)
Inits UpdateStatus with channles and config
Parameters
- **channels** (*list*) – A list of all of the ChannelWidget objects to be updated.
- **config** (*dict*) – Holds Cyckei launch settings.

run()
Goes through the channels list and sets the gui status text depending on server response from info_all query.

cyckei.client.workers.**prepare_json**(*channel*, *function*, *protocol*, *temp*)
Populates a new package with channel data and returns it

Parameters

- **channel** (ChannelWidget) – Channel object that stores info about itself.
- **function** (*str*) – Used by the server when determining what action to take.
- **protocol** (*str*) – A protocol for the server to execute.
- **temp** (*bool*) – True records in a temporary file, false records to the proper record directory.

Returns A package populated with the protocol and info about the specified channel.

Return type dict

7.3 Explorer

Main window for the cyckei client.

class cyckei.explorer.explorer.**MainWindow**(*config*)
Main Window class which is and sets up itself

__init__(*config*)
Setup main windows

cyckei.explorer.explorer.**main**(*config*)
Begins execution of Cyckei.

Parameters **record_dir** – Optional path to recording directory.

Returns Result of app.exec_(), Qt's main event loop.

Controls log tab, which displays logs as they are being recorded

class cyckei.explorer.log_viewer.**Folder**(*path*, *name*)
Object of log, stores title and content of file for quick access

class cyckei.explorer.log_viewer.**GraphCanvas**
Graphing Canvas using matplotlib

class cyckei.explorer.log_viewer.**Log**(*path*, *name*)
Object of log, stores title and content of file for quick access

class cyckei.explorer.log_viewer.**LogDisplay**

update(*self*) → None
update(self, arg__1: PySide2.QtCore.QRect) -> None update(self, arg__1: PySide2.QtGui.QRegion) -> None update(self, x: int, y: int, w: int, h: int) -> None

class cyckei.explorer.log_viewer.**LogViewer**(*config*, *resource*)
Object of log tab

log_clicked()
Display text of clicked file in text box

open_explorer(*text=None*)
Open logging folder in explorer

class cyckei.explorer.script_editor.**InsertBar**(*editor*)
Controls and stores information for a given channel

get_settings()
Creates all UI elements and adds them to elements list

update(*self*) → None
update(self, arg__1: PySide2.QtCore.QRect) -> None update(self, arg__1: PySide2.QtGui.QRegion) -> None update(self, x: int, y: int, w: int, h: int) -> None

class cyckei.explorer.script_editor.**Script**(*path*, *title*)
Stores the File Path, Title, and content of a file

save()
Overwrites the file that shares a file path and title witht the script

update_status()
Changes the file title in the Script interface if the script and the file differ

class cyckei.explorer.script_editor.**ScriptEditor**(*config*, *resource*)
UI window for the script tab of Cyckei Explorer

add(*file*)
Creates and adds a Script object to the front of the ScriptList and UI FileList

alert_check(*result*, *message*)
Opens a pop-up window with an input message

check(*text*)
Creates and runs a worker to check protocol and verify the validity of a script

delete(*text*)
Deletes the active file and removes it from the UI

help(*text*)
Opens the Cyclikal Guide for creating scripts

list_clicked()
Display contents of script when clicked

new(*text*)
Creates new file and adds it to list as script

open(*text*)
Opens a new file and adds it as a script

save(*text*)
Calls the save function of a Script object

setup_file_list()
Create list of script files

text_modified()
Update content of script and update status to show if edited

update_editor(*active_script_index*)
Updates the UI when which script is active is changed

class cyckei.explorer.workers.**Check**(*config*, *protocol*)

legal_test(*protocol*)
Checks if script only contains valid commands

run()
Initiates checking tests

class cyckei.explorer.workers.**Control**(*config*, *channel*, *command*, *script=None*, *temp=False*)
Update json and send "start" function to server

run(*self*) → None

class cyckei.explorer.workers.**Signals**

7.4 Server

Classes that handle controlling Keithley Source objects and enacting protocols on them.

class cyckei.server.protocols.**AdvanceCycle**(*wait_time: float = 10.0*, *cellrunner_parent: cyckei.server.protocols.CellRunner = None*)

Extends ProtocolStep. A step for advancing the cycle number in the parent CellRunner.

check_in_control(**args*)

Normally checks that battery is in control.

Since AdvanceCycle doesn't affect the connected cell True is always returned.

Returns Always returns True.

Return type bool

run(*force_report=False*)

Calls the parent CellRunner's advance_cycle() function.

Parameters **force_report** (*bool, optional*) – Forces a printed report if True. Unused in this case. Defaults to False.

class cyckei.server.protocols.**CCCharge**(*current*, *reports=(('voltage', 0.01), ('time', ':5:'))*, *ends=(('voltage', '>', 4.2), ('time', '>', '24::'))*, *wait_time=10.0*)

Extends CurrentStep. A step for enforcing a positive current.

__init__(*current*, *reports=(('voltage', 0.01), ('time', ':5:'))*, *ends=(('voltage', '>', 4.2), ('time', '>', '24::'))*, *wait_time=10.0*)

Inits state_str, calls the parent CurrentStep constructor with current, ends, reports, and wait_time.

Parameters

- **current** (*float*) – The current charge rate being enforced.
- **ends** (*tuple, optional*) – A tuple of tuples, holds the voltage cutoff and the total time the protocol should run for in hours:minutes:seconds format. Defaults to ((“voltage”, “<”, 3), (“time”, “>”, “24::”)).
- **reports** (*tuple, optional*) – A tuple of tuples, holds the change in voltage or time for a report to occur, time in in hours:minutes:seconds format. Defaults to ((“voltage”, 0.01), (“time”, “:5:”)).

- **wait_time** (*float, optional*) – Time between data measurements in seconds. Defaults to 10.0.

class cyckei.server.protocols.**CCDischarge**(*current*, *reports=(('voltage', 0.01), ('time', ':5:'))*, *ends=(('voltage', '<', 3), ('time', '>', '24::'))*, *wait_time=10.0*)

Extends CurrentStep. A step for enforcing a negative current.

__init__(*current*, *reports=(('voltage', 0.01), ('time', ':5:'))*, *ends=(('voltage', '<', 3), ('time', '>', '24::'))*, *wait_time=10.0*)

Inits state_str, calls the parent CurrentStep constructor with current, ends, reports, and wait_time.

Parameters

- **current** (*float*) – The current discharge rate being enforced.
- **ends** (*tuple, optional*) – A tuple of tuples, holds the voltage cutoff and the total time the protocol should run for in hours:minutes:seconds format. Defaults to ((“voltage”, “<”, 3), (“time”, “>”, “24::”)).
- **reports** (*tuple, optional*) – A tuple of tuples, holds the change in voltage or time for a report to occur, time in in hours:minutes:seconds format. Defaults to ((“voltage”, 0.01), (“time”, “:5:”)).
- **wait_time** (*float, optional*) – Time between data measurements in seconds. Defaults to 10.0.

class cyckei.server.protocols.**CVCharge**(*voltage*, *reports=(('current', 0.01), ('time', ':5:'))*, *ends=(('current', '<', 0.001), ('time', '>', '24::'))*, *wait_time=10.0*)

Extends VoltageStep. A step for charging at a constant voltage.

__init__(*voltage*, *reports=(('current', 0.01), ('time', ':5:'))*, *ends=(('current', '<', 0.001), ('time', '>', '24::'))*, *wait_time=10.0*)

[summary]

Parameters

- **ends** (*tuple, optional*) – A tuple of tuples, holds the current cutoff and the total time the protocol should run for in hours:minutes:seconds format. Defaults to ((“current”, “<”, 0.001), (“time”, “>”, “24::”)).

- **reports** (*tuple, optional*) – A tuple of tuples, holds the change in current or time for a report to occur, time in in hours:minutes:seconds format. Defaults to ((“current”, 0.01), (“time”, “:5:”)).
- **voltage** (*float*) – The desired voltage for the cell to reach.
- **wait_time** (*float, optional*) – Time between data measurements in seconds. Defaults to 10.0.

class cyckei.server.protocols.**CVDischarge**(*voltage*, *reports=(('current', 0.01), ('time', ':5:'))*, *ends=(('current', '<', 0.001), ('time', '>', '24::'))*, *wait_time=10.0*)

Extends VoltageStep. A step for discharging at a constant voltage.

__init__(*voltage*, *reports=(('current', 0.01), ('time', ':5:'))*, *ends=(('current', '<', 0.001), ('time', '>', '24::'))*, *wait_time=10.0*)

Inits state_str, calls Parent Class' constructor with voltage, reports, ends, and wait_time.

Parameters

- **ends** (*tuple, optional*) – A tuple of tuples, holds the current cutoff and the total time the protocol should run for in hours:minutes:seconds format. Defaults to ((“current”, “<”, 0.001), (“time”, “>”, “24::”)).
- **reports** (*tuple, optional*) – A tuple of tuples, holds the change in current or time for a report to occur, time in in hours:minutes:seconds format. Defaults to ((“current”, 0.01), (“time”, “:5:”)).
- **voltage** (*float*) – The desired voltage for the cell to reach.
- **wait_time** (*float, optional*) – Time between data measurements in seconds. Defaults to 10.0.

class cyckei.server.protocols.**CellRunner**(*plugin_objects=None*, ***meta*)

Turns a protocol into a list of held ProtocolSteps that are executed to complete the protocol. Also holds meta information about the protocol being run.

channel

The Keithley channel this protocol should be run on.

Type str

current_step

The active ProtocolStep. UNUSED.

Type *ProtocolStep*

fpath

The file path to the file that will have data written to it.

Type str

i_current_step
The index of the ProtocolStep being run from the steps list.
Type int

isTest
Controls whether this is a real protocol run or a test protocol being run.
Type bool

last_data
A list of values from the previous measurement recorded in a ProtocolStep.
Type list

meta
Meta data for: channel, path, cellid, comment, package, celltype, requester, plugins, protocol, protocol_name, cycler, start_cycle, and format.
Type dict

_next_time
The next time at which a ProtocolStep should read data from the Keithley.
Type float

plugin_objects
A list of PluginControllers extending the BaseController object. (The same as 'plugins' and 'plugin_objects' in functions of server.py)
Type list

prev_cycle
The previous cycle number. UNUSED.
Type int

safety_reset_seconds
The number of seconds before the Keithley's safety reset.
Type float

source
The Keithley being controlled by this CellRunner.
Type *keithley2602.Source*

start_time
The epoch time in seconds at which the CellRunenr started running the protocol (ProtocolSteps).
Type float

status
The status that maps to the STATUS string map. Values -1 to 5.
Type int

steps
A list of the ProtocolSteps to be run in order to complete a protocol.
Type list

total_pause_time
The time in seconds that a ProtoclStep has been paused for.
Type float

__init__ (*plugin_objects=None*, ***meta*)
Inits channel, current_step, fpath, i_current_step, last_data, meta, _next_time, plugin_objects, prev_cycle, safety_reset_seconds, source, start_time, status, steps, and total_pause_time.

Parameters **plugin_objects** (*list, optional*) – A list of PluginControllers extending the BaseController object. (The same as 'plugins' and 'plugin_objects' in functions of server.py) Defaults to None.

add_step (*step*)
Adds a ProtocolStep to the steps list.

Parameters **step** (ProtocolStep) – ProtocolStep to be added to the steps list.

advance_cycle ()
Advances the cycle stored in CellRunner by 1.

close ()
Calls the off() function for the stored source.

load_protocol (*protocol: str*, *isTest=False*)
Executes a string of python code to add a step to the steps list.

Parameters **protocol** (*str*) – string of python code generating the protocol steps in the runner.

next_step ()
Attempts to move to the next step of the protocol.

Attempts to increment the i_current_step by 1 to move to the next step in the steps list. If the length of steps is passed then False is returned and status is set to completed to indicate that the prtocol is over. Otherwise, capacity is adjusted to the last recorded capactiy and true is returned.

Returns True indicates that the next step is ready, False is returned if there are no more steps.

Return type bool

next_time
Returns the next time to read data.
Returns The next time in seconds at which to read data.
Return type float

off()
Calls the off() function for the stored source.

pause()
Attempts to pause the active step. Also calls the read_and_write function.
Returns True if the pause was successful, otherwise False.
Return type bool

read_and_write(*force_report=False*)
Reads data from the current ProtocolStep and records it.

Calls the current ProtocolStep to collect data and tries to write that data to the data file using write_data(). Also sets the last_data value to this most recently read data.
Parameters **force_report** (*bool, optional*) – Passed to the ProtocolStep run() function to control reporting. Defaults to False.

resume()
Calls resume on the current step. Then calls the run() function.
Returns The result of calling the run() function.
Return type bool

run(*force_report=False*)
Starts and advances protocols.

Method called by the main loop to start and advance a protocol This method triggers the various steps that are loaded in the protocol as well as the header writing and the data writing in the data file
Returns True if is running as expected, False if it is complete.
Return type bool

set_cap_signs(*direction=None*)

Decides whether charging/discharging yields positive or negative capacities.

Go through the steps and set their .cap_sign attribute based on the "direction" of the cell. Charging yields positive capacities because cap_sign is 1 and discharge yields negative capacities because cap_sign is -1.

Parameters **direction** (*str or None*) – Direction for the cell can be "pos", "neg", or None if it is None, the "celltype" in the meta data is used, if this is not set it defaults to "pos".

set_source(*source*)

Tries to set the CellRunner's source (the Keithley it controlls) to the passed in source.

Parameters **source** (keithley2602.Source) – The Source object used for controlling a specific Keithley.

Raises ValueError – Error raised when the CellRunner does not have the same channel as the Keithley it is controlling.

step

Uses the index of the current step to pull the current step from the steps list.

Returns The current step that the CellRunner is on.

Return type *ProtocolStep*

stop()

Calls the close() function.

Returns Always returns True.

Return type bool

write_cycle_header()

Writes the cycle that the CellRunner is on to the file stored in fpath.

write_data(*timestamp*, *current*, *voltage*, *capacity*, *plugin*)
Attempts to write the passed in data to the fpath file.

Parameters
- **capacity** (*float*) – The calculated capacity of the cell being controlled in mAh.
- **current** (*float*) – The recorded current data of the controlled cell.
- **plugin** (*list*) – A list of values recorded from plugins, either ints or floats.
- **timestamp** (*float*) – The recorded time data of the controlled cell.
- **voltage** (*float*) – The recorded voltage data of the controlled cell.

write_header()
Creates a JSON string using the CellRunner meta and writes it to the file stored in fpath.

write_step_header()
Collects the header from the current ProtocolStep and writes it to the file stored in fpath.

class cyckei.server.protocols.**Condition**
A Condition is an abstract class. A Condition takes a ProtocolStep into its check() method, and returns a boolean indicating whether that condition has been met.

A condition object may modify the .next_time attribute of the ProtocolStep object to suggest time the next time as an absolute timestamp that the condition should be checked

check(*protocol_step: cyckei.server.protocols.ProtocolStep*)
Abstract Function that checks whether certain conditions are met and returns a bool.

Parameters **protocol_step** (ProtocolStep) – The step being checked on whether it has met the condition.

Raises NotImplementedError – Raises this by default since check is an Abstract Method.

class cyckei.server.protocols.**ConditionAbsolute**(*value_str: str*, *operator_str: str*, *value: float*, *min_time: float = 1.0*)
An object for comparing the most recent measurement of a ProtocolStep to a user designated value.

comparison
The comparison func to use when comparing values i.e. greater than, less than, etc.
Type func

index
The index that relates to the value string in the data lists from Steps.
Type int

min_time
Minimum time that must have elapsed before evaluating the condition.
Type float

next_time
The next expected time for data to be checked. NEVER USED. Defaults to infinity.
Type float

value
Actual value to compare against step data.
Type float

value_str
Value string such as "voltage", "time", "current" etc... See DATA_INDEX_MAP module variable for valid values.
Type str

__init__(*value_str: str*, *operator_str: str*, *value: float*, *min_time: float = 1.0*)
Inits with comparison, index, min_time, next_time, value, and value_str.
Parameters
- **value_str** (*str*) – Value string such as "voltage", "time", "current" etc... See DATA_INDEX_MAP module variable for valid values.
- **operator_str** (*str*) – String indicating the comparison operator. See OPERATOR_MAP module variable for valid values.
- **value** (*float*) – Actual value to compare against step data.
- **min_time** (*float*) – Minimum time that must have elapsed before evaluating the condition.

check(*step*)
Compares absolute set value to step data.

First checks to see if enough time has passed between the most recent report and the newest data. If enough time has passed then the newest data value is compared against the set value.
Parameters **step** (ProtocolStep) – The step to have data pulled from its data list.
Returns True if the end condition was satisfied, otherwise False
Return type bool

class cyckei.server.protocols.**ConditionDelta**(*value_str: str*, *delta: float*)

Condition that checks change between latest reported value and latest measured value.

comparison

An operator function that performs comparisons, in this case it is greater than or equal to, since the comparison is change in value.

Type func

delta

Delta to compare step data against.

Type float

index

The integer that maps to a constant data map referencing data type being compared.

Type int

is_time

True means that this condition compares time values. False means it does not.

Type bool

value_str

Value string such as "voltage", "time", "current" etc... See DATA_INDEX_MAP module variable for valid values.

Type str

__init__(*value_str: str*, *delta: float*)

Inits with comparison, delta, index, and is_time, and value_str.

Parameters

- **value_str** (*str*) – Value string such as "voltage", "time", "current" etc... See DATA_INDEX_MAP module variable for valid values.
- **delta** (*float*) – Delta to compare step data against.

check(*step*)

Checks the provided step's value against the delta value.

Takes the most recent list of readings from the data list in step and indexes into the matching value to compare against delta.

Parameters **step** (ProtocolStep) – The step to have data pulled from its data list.

Returns True if the end condition was satisfied, otherwise False

Return type bool

class cyckei.server.protocols.**ConditionTotalDelta**(*value_str: str*, *delta: float*)

Object for checking the change between the first reported value and latest measured value of a step.

comparison
An operator function that performs comparisons, in this case it is greater than or equal to, since the comparison is change in value.
Type func

delta
Delta to compare step data against.
Type float

index
The integer that maps to a constant data map referencing data type being compared.
Type int

next_time
At what timestamp do we expect the condition to be met. Defaults as infinite.
Type float

value_str
Value string such as "voltage", "time", "current" etc... See DATA_INDEX_MAP module variable for valid values.
Type str

__init__(*value_str: str*, *delta: float*)
Inits with comparison, delta, index, next_time, and value_str.
Parameters
- **value_str** (*str*) – Value string such as "voltage", "time", "current" etc... See DATA_INDEX_MAP module variable for valid values.
- **delta** (*float*) – Delta to compare step data against.

check(*step*)
Checks the provided step's value against the delta value.

Takes the first list of readings from the data list in step and indexes into the matching value to compare against delta.
Parameters **step** (ProtocolStep) – The step to have data pulled from its data list.
Returns True if the end condition was satisfied, otherwise False.
Return type bool

class cyckei.server.protocols.**ConditionTotalTime**(*delta*)
Object for checking the change between the first reported time and latest measured time of a step.

Extends ConditionTotalDelta and simply calls its parent class' constructor with time as the value_str.

__init__(*delta*)

Calls the parent class' constructor to make a ConditionTotalDelta with time.

Parameters **delta** (*float*) – Total elapsed time in seconds.

check(*step*)

Checks the provided step's value against the delta value.

Takes the first list of readings from the data list in step and indexes into the matching value to compare against delta.

Parameters **step** (ProtocolStep) – The step to have data pulled from its data list.

Returns True if the end condition was satisfied, otherwise False.

Return type bool

class cyckei.server.protocols.**CurrentStep**(*current*, *reports=(('voltage', 0.01), ('time', ':5:'))*, *ends=(('voltage', '>', 4.2), ('time', '>', '24::'))*, *wait_time=10.0*)

Extends ProtocolStep. A step for controlling the current output by the source (i.e Keithley).

current

The current rate being enforced.

Type float

v_limit

Voltage limit for source. This is not a voltage cutoff condition. It is the maximum voltage allowed by the Keithley under any condition. The Keithley enforces +/- v_limit. Having a battery with a voltage outside of +/- v_limit could damage the Keithley. Defaults to 5.0.

Type float

__init__(*current*, *reports=(('voltage', 0.01), ('time', ':5:'))*, *ends=(('voltage', '>', 4.2), ('time', '>', '24::'))*, *wait_time=10.0*)

Inits current, end_conditions, report_conditions, state_str, and v_limit. Calls parent ProtocolStep constructor with wait_time.

Parameters

- **current** (*float*) – The current rate being enforced.
- **ends** (*tuple, optional*) – A tuple of tuples, holds the voltage cutoff and the total time the protocol should run for in hours:minutes:seconds format. Defaults to (("voltage", ">", 4.2), ("time", ">", "24::")).

- **reports** (*tuple, optional*) – A tuple of tuples, holds the change in voltage or time for a report to occur, time in in hours:minutes:seconds format. Defaults to ((“voltage”, 0.01), (“time”, “:5:”)).
- **wait_time** (*float, optional*) – Time between data measurements in seconds. Defaults to 10.0.

Raises ValueError – Current should not be 0 during a CurrentStep, this is raised if current is 0.

check_in_control(*last_time*, *current*, *voltage*)

Method for checking if the desired condition is actually met.

Returning False will completely kill the cell.

Parameters

- **current** (*float*) – The current rate being enforced.
- **last_time** (*float*) – The measured time in seconds. UNUSED
- **voltage** (*float*) – The measured voltage to be compared against this step’s set voltage. UNUSED

Returns True if cell is in control, False otherwise

Return type bool

header()

Returns the current state and time in json form.

Returns A JSON string of the current protocol state and time.

Return type JSON

class cyckei.server.protocols.**Pause**

Extends ProtocolStep. Pauses the CellRunner and its Keithley source.

__init__()

Inits state_str and invokes the parent’s constructor with infinite wait time.

check_in_control(**args*)

When a everything is paused in_control means nothing, hence this function does nothing but return True.

Returns Always returns True
Return type bool

resume()
Sets the Pause step's status to completed in order to move on from being paused.

run()
Calls the Pause start() function.
Returns None returned.
Return type None

class cyckei.server.protocols.**ProtocolStep**(*wait_time: float = 10.0*, *cellrunner_parent: cyckei.server.protocols.CellRunner = None*)
Base class for a protocol step, needs to be subclassed with implementation of a start function.

A protocol step stores its own data and the reported points Keeps track of time, current, voltage, capacity. Because the protocol files are simply pure python the parent, which is a CellRunner instance, needs to be present in the global variables as "parent".

cap_sign
The cap sign determines whether the capacitiy increases or decreases during charge and discharge. Either 1 or -1.
Type float

data
A list of lists. Each list is a set of measurements, [[time,current,voltage,capacity], [time,current,voltage,capacity], ...] current is stored in absolute value. Contains every measurement taken.
Type list

data_max_len
The max number ofitems in the data list.
Type int

end_conditions
A list of conditions that determine when the ProtocolStep should be ended.
Type list

in_control
Indicates if the protocol step is operating within it's designed parameters.
Type bool

last_time
The previous measured time in seconds.

Type float

next_time
This is the time in seconds since epoch at which this protocol step is expecting to do another read operation on the channel. -1 means it has never been set.
Type float

parent
The CellRunner holding this protocol step.
Type *CellRunner*

pause_start
The time in seconds at which the protocol was paused.
Type float

pause_time
The total time in seconds for which the protocol was paused.
Type float

report
A list of lists. Each list is a set of measurements, [[time,current,voltage,capacity], [time,current,voltage,capacity], . . .]. This list only contains measurements taken that are specified to be reported.
Type list

report_conditions
A list of Condition objects used to determine when a data measurement is reported.
Type list

starting_capacity
The initital capacity of the cell. This gets set at the protocol level (parent).
Type float

state_str
A string representation of the state of the cell i.e charging, discharging, etc.
Type str

status
An int representation of the status of the step, i.e started, paused, etc.
Type int

wait_time
Time between data measurements in seconds.
Type float

__init__(*wait_time: float = 10.0*, *cellrunner_parent: cyckei.server.protocols.CellRunner = None*)
Inits ProtocolStep with parent, data_max_len, status, state_str, last_time, pause_start, pause_time, cap_sign, next_time, starting_capacity, wait_time, end_conditions, report_conditions, in_control.

Base class for protocols the variable "parent" must be a CellRunner instance and be present in the globals during instantiation.
Parameters
- **cellrunner_parent** (CellRunner) – The CellRunner this protocol is attached to.

- **wait_time** (*float*) – Default waiting time in seconds. If no other conditions are met, the step will check V & I at this interval.

check_end_conditions()
Checks if it's time for step to be ended.
Returns True if the end condition was satisfied, otherwise False
Return type bool

check_in_control(*last_time*, *current*, *voltage*)
Abstract Method for checking if the desired condition is actually met.
For example
- constant current : check current is correct
- constant voltage: check voltage is correct

Parameters
- **current** (*float*) – Current in amps. NOTE THAT THIS IS NOT ABSOLUTE CURRENT IT IS THE CURRENT DIRECTLY REPORTED FROM THE INSTRUMENT.
- **last_time** (*float*) – Timestamp of last measurement.
- **voltage** (*float*) – Voltage in volts.

Raises NotImplementedError – Raises immediately as this is an Abstract Method.

check_report_conditions()
Check if it's time for step info to be reported.
Returns True if the end condition was satisfied, otherwise False
Return type bool

header()
Unimplemented function. Meant for being overridden.
Returns Always returns False.
Return type bool

pause()
If step is started turns the source off and sets the status to paused.
Returns Returns False if the step hasn't been started. Otherwise True.
Return type bool

read_data(*force_report=False*)
Reads and reports data from the Keithley source.

Reads data from the Keithley source. Next scans the list of plugins checking for active ones. If there are active plugins their read() function is called and the output is checked. int and float are acceptable return values from the plugins. A 0 will replace non int or non float values. If data has been previously reported that data is used to calculate current capacity, otherwise capacity is the starting capacity. The read values of last_time, current, voltage, capacity, plugin_values are then compiled into a list and appended to the data list. If the data list is oversized its second to last oldest value is popped from the list (removing the first value would mess up calculating total changes). Finally the data is added to the end of the report if force_report is true.
Parameters **force_report** (*bool, optional*) – If True then the collected data is added to the report list. Defaults to False.

resume()
Resumes the step by calling _start(), also calculates pause time.
Returns Returns True if the step hasn't been paused.
Return type bool

run(*force_report=False*)
Calls the read_data() function, also decides if the read data should be reported.
Calls the read_data() function and checks the end_conditions for if the ProtocolStep should end. The run function evaluates the report_conditions to decide if a data point should be reported, setting force_report to True will report the latest data point regardless of conditions.

Parameters **force_report** (*bool*) – Defaults to False. Forces a report if True.
Returns
(time, current, voltage, capacity) tuple to report (write to file). Returns none if no data to report.
Return type tuple

class cyckei.server.protocols.**Rest**(*reports=(('time', ':5:'),), ends=(('time', '>', '24::'),), wait_time=10.0*)

Extends ProtocolStep. A step for putting a the CellRunner and Keithley to rest.

__init__(*reports=(('time', ':5:'),), ends=(('time', '>', '24::'),), wait_time=10.0*)

Inits end_conditions, report_conditions, and state_str.

Parameters

- **ends** (*tuple, optional*) – The total time the protocol should run for in hours:minutes:seconds format. Defaults to ((“time”, “>”, “24::”),).
- **reports** (*tuple, optional*) – The time betweem reports in hours:minutes:seconds format. Defaults to ((“time”, “:5:”),).
- **wait_time** (*float, optional*) – Time between data measurements in seconds. Defaults to 10.0.

check_in_control(*last_time*, *current*, *voltage*)

Method for checking if the desired condition is actually met.

Compares the given current to 0.00001 to check if it is essentially 0. Sets in_control, the return value, to True if the current is essentially 0. Returning False will completely kill the cell.

Parameters

- **current** (*float*) – The measured current to compare to 0.00001.
- **last_time** (*float*) – The last measurement time, UNUSED.
- **voltage** (*float*) – The measured voltage of the cell, UNUSED.

Returns True if cell is in control, False otherwise.

Return type bool

header()

Returns the current state and time in json form.

Returns A JSON string of the current protocol state and time.

Return type JSON

class cyckei.server.protocols.**Sleep**(*reports=(('time', ':5:'),), ends=(('time', '>', '24::'),), wait_time=10.0*)

Extends ProtocolStep. A protocol used for putting the CellRunner and Keithley to sleep.

__init__(*reports=(('time', ':5:'),), ends=(('time', '>', '24::'),), wait_time=10.0*)
Inits end_conditions, report_conditions, and state_str.

Parameters
- **ends** (*tuple, optional*) – The total time the protocol should run for in hours:minutes:seconds format. Defaults to (("time", ">", "24::"),).
- **reports** (*tuple, optional*) – The time betweem reports in hours:minutes:seconds format. Defaults to (("time", ":5:"),).
- **wait_time** (*float, optional*) – Time between data measurements in seconds. Defaults to 10.0.

check_in_control(*last_time*, *current*, *voltage*)
Method for checking if the desired condition is actually met.

Returning False will completely kill the cell

Parameters
- **()** (*voltage*) –
- **()** –
- **()** –

Returns True if cell is in control, False otherwise
Return type bool

header()
Returns the current state and time in json form.

Returns A JSON string of the current protocol state and time.
Return type JSON

read_data()
Reads the data from the Keithley source by calling the parent class' read_data().

run(*force_report=False*)
Calls the start function of the sleep protocol, checks end conditions, and reports data.

The run method needs to be redefined because the logic of the Sleep protocol is unique in that a measurement is only desired if a report condition is met as opposed to the other steps which measure and then decide whether to report

Parameters **force_report** (*bool*) – Defaults to False. The run function evaluates the report_conditions to decide if a data point should be reported, setting force_report to True will report the latest data point regardless of conditions.
Returns

(time, current, voltage, capacity) tuple to report (write to file). Returns none if no data to report.

Return type tuple

class cyckei.server.protocols.**VoltageStep**(*voltage*, *reports=(('current', 0.01), ('time', ':5:'))*, *ends=(('current', '<', 0.001), ('time', '>', '24::'))*, *wait_time=10.0*)

Extends ProtocolStep. A step for controlling the voltage of a cell.

i_limit

The maximum allowed current when charging or discharing.

Type float

voltage

The desired voltage for the cell to reach.

Type float

__init__(*voltage*, *reports=(('current', 0.01), ('time', ':5:'))*, *ends=(('current', '<', 0.001), ('time', '>', '24::'))*, *wait_time=10.0*)

Inits i_limit, end_conditions, report_conditions, and voltage.

Parameters

- **ends** (*tuple, optional*) – A tuple of tuples, holds the current cutoff and the total time the protocol should run for in hours:minutes:seconds format. Defaults to ((“current”, “<”, 0.001), (“time”, “>”, “24::”)).
- **reports** (*tuple, optional*) – A tuple of tuples, holds the change in current or time for a report to occur, time in in hours:minutes:seconds format. Defaults to ((“current”, 0.01), (“time”, “:5:”)).
- **voltage** (*float*) – The desired voltage for the cell to reach.
- **wait_time** (*float, optional*) – Time between data measurements in seconds. Defaults to 10.0.

check_in_control(*last_time*, *current*, *voltage*)

Method for checking if the desired condition is actually met.

Voltage can take a moment to stabalize when a cell is first started, so the tolerance is adjusted accordingly. Otherwise, The currently measured voltage is compared against the voltage set in this step and if it is too different the in_control value is set to False. in_control is then returned. Returning False will completely kill the cell.

Parameters

- **current** (*float*) – The measured current. UNUSED.
- **last_time** (*float*) – The measured time in seconds.
- **voltage** (*float*) – The measured voltage to be compared against this step’s set voltage.

Returns True if cell is in control, False otherwise
Return type bool

guess_i_limit()
Takes the last value in the CellRunner Parent's Keithley's current_ranges and sets it positive or negative depending on charge or discharge.

header()
Returns the current state and time in json form.
Returns A JSON string of the current protocol state and time.
Return type JSON

cyckei.server.protocols.**condition_dc**(*dc*)
Function for creating a ConditionDelta object using capacity.

Parameters **dc** (*float*) – The change in capacity.

Returns A Condition that can be used on steps to compare changes in capacity.

Return type *ConditionDelta*

cyckei.server.protocols.**condition_di**(*di*)
Function for creating a ConditionDelta object using current.

Parameters **di** (*float*) – The change in current.

Returns A Condition that can be used on steps to compare changes in current.

Return type *ConditionDelta*

cyckei.server.protocols.**condition_dt**(*dt*)
Function for creating a ConditionDelta object using time.

Parameters **dt** (*float*) – The change in time in seconds.

Returns A Condition that can be used on steps to compare changes in time.

Return type *ConditionDelta*

cyckei.server.protocols.**condition_dv**(*dv*)
Function for creating a ConditionDelta object using voltage.

Parameters **dv** (*float*) – The change in voltage.

Returns A Condition that can be used on steps to compare changes in voltage.

Return type *ConditionDelta*

cyckei.server.protocols.**condition_end_voltage**(*voltage*, *operator_str*)
Function for creating a ConditionAbsolute object using a voltage endpoint.

Loads a ConditionAbsolute object with a comparison voltage and the mathematical comparison operator to use.

Parameters

- **voltage** (*float*) – The voltage to initialize ConditionAbsolute with.
- **operator_str** (*str*) – A string representing a mathematical operator i.e. ><= .

Returns Initialized with the given voltage and operator_str.

Return type *ConditionAbsolute*

cyckei.server.protocols.**condition_lcv**(*voltage*)
Function for creating a ConditionAbsolute object using a lower cutoff voltage.

Parameters **voltage** (*float*) – The voltage to initialize ConditionAbsolute with.

Returns Initialized with the given voltage and <= as the operator.

Return type *ConditionAbsolute*

cyckei.server.protocols.**condition_max_current**(*current*)
Function for creating a ConditionAbsolute object using a max current.

Parameters **current** (*float*) – The current to initialize ConditionAbsolute with.

Returns Initialized with the given current and >= as the operator.

Return type *ConditionAbsolute*

cyckei.server.protocols.**condition_min_current**(*current*)
Function for creating a ConditionAbsolute object using a min current.

Parameters **current** (*float*) – The current to initialize ConditionAbsolute with.

Returns Initialized with the given current and <= as the operator.

Return type *ConditionAbsolute*

cyckei.server.protocols.**condition_total_time**(*total_time*)
Function for creating a ConditionTotalTime object for a total time condition.

Parameters **total_time** (*float*) – The total_time to be used in the ConditionTotalTime.

Returns Initialized with the given total_time.

Return type *ConditionTotalTime*

cyckei.server.protocols.**condition_ucv**(*voltage*)
Function for creating a ConditionAbsolute object using an upper cutoff voltage.

Parameters **voltage** (*float*) – The voltage to initialize ConditionAbsolute with.

Returns Initialized with the given voltage and >= as the operator.

Return type *ConditionAbsolute*

cyckei.server.protocols.**extrapolate_time**(*data*, *target*, *index*)
Estimates the time until a voltage cutoff is reached.

Parameters

- **data** (*list*) – A list of lists of measurements taken (voltages, currents, times) at each time.
- **target** (*float*) – The target voltage being extrapolated to.
- **index** (*int*) – Which of the values being tested against i.e. [0:self.last_time, 1:current, 2:voltage, 3:capacity, 4:plugin_values]

Returns The time at which the target will be hit.

Return type float

cyckei.server.protocols.**process_ends**(*ends*)
Takes end_conditions in tuple form and converts them into Condition objects.

Parameters **ends** (*((str, str, float), (str, str, str))*) – Desired ends conditions as a tuple of triples such as ((“voltage”,”>”, 4.2), (“time”,”>”,”24::”)).

Returns List of Condition objects for ending a protocol step.

Return type list

cyckei.server.protocols.**process_reports**(*reports*)
Takes reports in the form of a tuple of pairs and creates a list of Condition objects for when to report.

Parameters **reports** (*((str, float), (str, int))*) – Desired reports as a tuple of pairs such as ((“voltage”,0.01), (“time”,300))

Returns List of condition objects for reporting a data point

Return type list

cyckei.server.protocols.**time_conversion**(*t*)
Converts time in the “hh:mm:ss” format to seconds as a float.

Parameters **t** (*str or float*) – time in the “hh:mm:ss” format, where values can be ommitted e.g. “::5” would be five seconds or time in seconds.

Returns Calculated time in Seconds.

Return type float

Main script run by server application

cyckei.server.server.**event_loop**(*config*, *socket*, *plugins*, *plugin_names*, *device_module*)
Main start method and loop for server application.

Connects cycling channels, sets up CellRunners for controlling channels, waits for runners then processes and discards them as necessary. Also records the channel statuses.

Parameters

- **config** (*dict*) – Holds Cyckei launch settings.
- **device_module** (*module*) – A module (in this case keithley.py) that includes a definition for DeviceController(gpib_addr (int)).
- **plugins** (*list*) – A list of PluginControllers extending the BaseController object.

- **plugin_names** (*dict*) – A dict with a key of the plugin name and a value of the of the specific plugin instance's name.
- **socket** (*zmq.Socket*) – An object that acts as a socket that can send and receive messages.

cyckei.server.server.**get_runner_by_channel**(*channel*, *runners*, *status=None*)

Get runner currently on given channel.

Parameters

- **channel** (*int or str*) – The channel number associated with the desired Keithley.
- **runners** (*list*) – A sorted list of active CellRunner objects.
- **status** (*int, optional*) – The status number associated with different runner statuses. -1 to 5. Defaults to None.

Returns Returns the runner serving the given channel, returns None otherwise.

Return type CellRunnner

cyckei.server.server.**info_all_channels**(*runners*, *sources*)

Return info on all channels

Parameters

- **runners** (*list*) – A sorted list of active CellRunner objects.
- **sources** (*list*) – A list of all of the Keithley channels connected to the server.

Returns A dictionary of dictionaries that each hold info from their respective CellRunner's meta, e.g. path, status, current, voltage, etc.

Return type dict

cyckei.server.server.**info_channel**(*channel*, *runners*, *sources*)

Return info about the specified channel.

Parameters

- **channel** (*int*) – The channel number associated with the desired Keithley.
- **runners** (*list*) – A sorted list of active CellRunner objects.
- **sources** (*list*) – A list of all of the Keithley channels connected to the server.

Returns Information about the requested channel from the CellRunner's meta, e.g. path, status, current, voltage, etc.

Return type dict

cyckei.server.server.**info_server_file**(*config*)
Return the dict of channels in the server file

Parameters **config** (*dict*) – Holds Cyckei launch settings.

Returns The json data of channels recorded in a file, converted to a dict.

Return type dict

cyckei.server.server.**main**(*config*, *plugins*, *plugin_names*)
Begins execution of Cyckei Server.

Sets up the socket for communication with a client and starts the event_loop to process commands.

Parameters

- **config** (*dict*) – Holds Cyckei launch settings.
- **plugins** (*list*) – A list of PluginControllers extending the BaseController object.
- **plugin_names** (*dict*) – A dict with a key of the plugin name and a value of the of the specific plugin instance's name.

cyckei.server.server.**pause**(*channel*, *runners*)
Pauses the specified channel.

Parameters

- **channel** (*int*) – The channel number associated with the desired Keithley.
- **runners** (*list*) – A sorted list of active CellRunner objects.

Returns The result message of trying to pause a channel.

Return type str

cyckei.server.server.**process_socket**(*config*, *socket*, *runners*, *sources*, *server_time*, *plugins*, *plugin_names*)
Checks the running socket for messages and then parses them into actions to take.

Parameters

- **config** (*dict*) – Holds Cyckei launch settings.

- **plugins** (*list*) – A list of PluginControllers extending the BaseController object.
- **plugin_names** (*dict*) – A dict with a key of the plugin name and a value of the of the specific plugin instance's name.
- **runners** (*list*) – A sorted list of active CellRunner objects.
- **server_time** (*float*) – The time on the server (unused in the function)
- **socket** (*zmq.REP socket*) – Receives messages in a non-blocking way. If a message is received it processes it and sends a response
- **sources** (*list*) – A list of all of the Keithley channels connected to the server.

cyckei.server.server.**record_data**(*data_path*, *data*)

Saves server status to a file.

Uses the data_path to open an existing or new file, converts the data to a json, then writes the data to the file. If there is already existing data in channels it is not overwritten by the same channels now being empty.

Parameters

- **data** (*dict*) – The data to be stored in a file.
- **data_path** (*str*) – The path to the area where the user wants the server_data file stored.

cyckei.server.server.**resume**(*channel*, *runners*)

Attempts to resume the specified channel from pause.

Parameters

- **channel** (*int*) – The channel number associated with the desired Keithley.
- **runners** (*list*) – A sorted list of active CellRunner objects.

Returns The result message of trying to resume a channel.

Return type str

cyckei.server.server.**start**(*channel*, *meta*, *protocol*, *runners*, *sources*, *plugin_objects*)

Start channel with given protocol.

Parameters

- **channel** (*int*) – The channel number associated with the desired Keithley.
- **meta** (*dict*) – The metadata about a channel, which is provided to the CellRunner.

- **plugin_objects** (*list*) – A list of PluginControllers extending the BaseController object. (The same as 'plugins' in other functions of server.py)
- **protocol** (*str*) – The protocol to be loaded onto a CellRunner.
- **runners** (*list*) – A sorted list of active CellRunner objects.
- **sources** (*list*) – A list of all of the Keithley channels connected to the server.

Returns The result message of trying to start a channel.

Return type str

cyckei.server.server.**stop**(*channel*, *runners*)

Stop the specified channel.

Parameters

- **channel** (*int*) – The channel number associated with the desired Keithley.
- **runners** (*list*) – A sorted list of active CellRunner objects.

Returns The result message of trying to strop a channel.

Return type str

cyckei.server.server.**test**(*protocol*)

Test the specified protocol for compliance.

Parameters **()** (*protocol*) –

Returns The result message of testing the protocol.

Return type str

Classes to handle interfacing with Keithleys and their channels

class cyckei.server.keithley2602.**DeviceController**(*gpib_addr*, *load_scripts=True*, *safety_reset_seconds=120*)

Represents a single keithley Interface

gpib_addr

Either the int part of the GPIB address or the full GPIB address as a str.

Type int or str

safety_reset_seconds

How many seconds the Keithley can go without being checked before being shut off.

Type int

source_meter
The Keithley connected using pyvisa.
Type visa GPIBInstrument

__init__(*gpib_addr*, *load_scripts=True*, *safety_reset_seconds=120*)
Inits Device Controller with gpib_addr, safety_reset_seconds, and source_meter.

Also resets the source meter and initializes it with either a startup scrip or a safety shut off script.
Parameters
- **gpib_addr** (*int or str*) – Either the int part of the GPIB address or the full GPIB address as a str.
- **load_scripts** (*bool, optional*) – Defaults to True. Whether the source should be able to load scripts.
- **safety_reset_seconds** (*int, optional*) – How many seconds the Keithley can go without being checked before being shut off.

get_source(*kch*, *channel=None*)
Creates a source object of a Keithley with the specified kch channel.
Parameters
- **kch** (*str*) – 'a' or 'b', used to set whether a or b on the keithley is used.
- **channel** (*str, optional*) – User specified name of the channel. Defaults to None

Returns Initialized with source_meter, kch, channl, and safety_reset_seconds.
Return type *Source*

class cyckei.server.keithley2602.**Source**(*source_meter*, *kch*, *channel=None*, *safety_reset_seconds=120*)
Represents an individual source.

Typically generated from a Keithley object's get_source function

channel
Channel name that the user sees. Will be used to sort and display channels. Defaults to None.
Type str

chd
Holds the connection between kch and snum, {"a":1, "b":2}.
Type dict

data
The backlog of data being stored; holds timestamp, current, and voltage.
Type list

data_max_len
Defaults to 500. The legnth of the backlog of data being stored.
Type int

identification
Either 'smua' or 'smub', corresponds with whether the kch is 'a' or 'b'.
Type str

kch
Keithley channel ("a" or "b"). Stored lower case internally and accessible in the kch attribute.
Type str

report
List of points from data list added whenever a condition is met.
Type list

safety_reset_seconds
How many seconds the Keithley can go without being checked before being shut off.
Type int

source_meter
The Keithley source, btained from an open_resource pyvisa call.
Type visa GPIBInstrument

snum
Either 1 or 2, corresponds with whether the kch is 'a' or 'b'.
Type int

__init__(*source_meter*, *kch*, *channel=None*, *safety_reset_seconds=120*)
Constructs necessary parameters for a Source object.
Parameters
- **channel** (*int or str*) – Channel that the user sees. Can be integer or string, however will be used to sort and display channels. Defaults to None.
- **kch** (*str*) – Keithley channel ("a" or "b"). Stored lower case internally and accessible in the kch attribute.
- **source_meter** (*visa GPIBInstrument*) – The Keithley source, btained from an open_resource pyvisa call.

get_range(*current*)
Compares the current_range to the provided current and returns the smallest number in the range still larger than the provided current.
Parameters **current** (*int*) – Current to compare to the current_range
Returns
The smallest current in the current_range still larger than the provided current.
Return type int

off()
Stops the protocol on the Keithley and sets the Keithley to off-mode.

pause()
Attempts to pause the Keithley script by writing abort to the Keithley and changing the output.

query(**args*, ***kwargs*)
Makes a call to query the Keithley source.

read_data(**args*, ***kwargs*)
Shuts the Keithely off after {safety_reset_seconds} if the Keithley is not checked in that time.
Returns The result of the function that is being called through here. Could return anything.
Return type Any

read_iv(**args*, ***kwargs*)
Shuts the Keithely off after {safety_reset_seconds} if the Keithley is not checked in that time.
Returns The result of the function that is being called through here. Could return anything.
Return type Any

read_until(*write_conditions*, *end_conditions*, *wait_time=5.0*)
Records data from a Keithley at regular intervals until an end condition is met.
Parameters
- **end_conditions** (*list*) – A list of Conditions from protocols.py to be checked against to end the process.
- **wait_time** (*int, optional*) – Time in seconds between checks. Defaults to 5.
- **write_conditions** (*list*) – A list of Conditions from protocols.py to be checked against determining whether to write data.

rest(**args*, ***kwargs*)
Shuts the Keithely off after {safety_reset_seconds} if the Keithley is not checked in that time.
Returns The result of the function that is being called through here. Could return anything.
Return type Any

set_current(**args*, ***kwargs*)
Shuts the Keithely off after {safety_reset_seconds} if the Keithley is not checked in that time.
Returns The result of the function that is being called through here. Could return anything.
Return type Any

set_text(*text1: str = ''*, *text2: str = ''*)
Sets the display test on the Keithley screen

Unfortunately the Keithley does not treat the two channels independently for display purposes. So setting the text for one channels removes all the info for the other channel rendering this functionality nearly useless.
Parameters
- **text1** (*str*) – top line of text, 10 max chars
- **text2** (*str*) – bottom line of text, 16 max chars

set_voltage(**args*, ***kwargs*)
Shuts the Keithely off after {safety_reset_seconds} if the Keithley is not checked in that time.
Returns The result of the function that is being called through here. Could return anything.
Return type Any

write(**args*, ***kwargs*)
Makes a call to write the included arguments to the Keithley source.

cyckei.server.keithley2602.**parse_gpib_address**(*gpib_address*)
Takes int or str and returns full str GPIB address

Parameters **gpib_address** (*int or str*) – Either the int part of the GPIB address or the full GPIB address as a str.

Returns The full GPIB address.

Return type str

cyckei.server.keithley2602.**with_safety**(*fn*)

Wrapper function for the Source class to enforce the use of a safety script

Safety cutoff will shut the keithley off after {safety_reset_seconds} if it is not at least checked.

Parameters **fn** (*funtion*) – The function being decorate for safety cutoff.

Returns The result of the function that is being called through here. Could return anything.

Return type Any

7.5 Functions

cyckei.functions.func.**asset_path**(*path*)

Appends a given path to the end of the path to the assests folder.

Parameters **path** (*str*) – The path starting from the assests folder.

Raises FileNotFoundError – Error is raised when the given path doesn't point to an existing folder or file.

Returns the input path appended to the assests folder path.

Return type str

cyckei.functions.func.**not_none**(*value*)

Sets a None value to "None" string

Parameters **value** (*None*) – Expects a None, but able to handle anything convertabile to a str.

Returns Returns "None" as a string or converts the given value to a string and returns it.

Return type str

Universal GUI Functions

cyckei.functions.gui.**action**(*text=None*, *connect=None*, *tip=None*, *parent=None*, *disabled=False*, *separator=False*)

Creates an "action" that can be linked to the other ui elements in order to perform functions.

Uses the QAction object to implement this functionality.

Parameters

- **text** (*str, optional*) – The descriptive text displayed by the action. Defaults to None.
- **connect** (*func, optional*) – The function that is called when the action is triggered. Defaults to None.
- **tip** (*str, optional*) – The status tip is displayed on all status bars provided by the action's top-level parent. Defaults to None.
- **parent** (*QObject, optional*) – If parent is an action group the action will be automatically inserted into the group. Defaults to None.
- **disabled** (*bool, optional*) – Whether the action is disabled or not, True means the action is inactive (disabled). Defaults to False.
- **separator** (*bool, optional*) – True makes this action a separator, an action that physically separates other actions in the GUI. Defaults to False.

Returns The action object that can be connected to the other gui elements.

Return type QAction

cyckei.functions.gui.**button**(*text=None*, *status=None*, *connect=None*, *enabled=True*)

Creates a QPushButton with given information.

Parameters

- **text** (*str, optional*) – The text on the button. Defaults to None.
- **status** (*str, optional*) – The text of the status tip that appears when the cursor hovers over the button. Defaults to None.
- **connect** (*func, optional*) – The function to connect to pushing the button. Defaults to None.
- **enabled** (*bool, optional*) – Whether the button is enabled or not. False is Disabled. Defaults to True.

Returns A button with the specified features.

Return type QPushButton

cyckei.functions.gui.**combo_box**(*items*, *status*, *key*, *connect*)
Creates a QComboBox with given information. Essentially creates a dropdown selector.

This is a combo box that is connected to a specified function.

Parameters

- **items** (*list*) – Adds a list of strings as selectable items to the dropdown.
- **status** (*str*) – The text of the status tip that appears when the cursor hovers over the button.
- **key** – The parameter that will be needed by the connected function (connect).
- **connect** (*func*) – The function to connect to selecting an item in the dropdown window.

Returns A combined button and popup list with the specified features.

Return type QComboBox

cyckei.functions.gui.**feedback**(*status*, *channel*)
Changes the text in the feedback label object in the provided channel.

Parameters

- **status** (*str*) – The string to be diplayed as the status of the channel.
- **channel** (ChannelWidget) – The channel to have its feedback label edited.

cyckei.functions.gui.**label**(*text*, *status=None*, *tag=None*)
Creates a QLabel for your QApplication.

Parameters

- **text** (*str*) – Sets the text of the QLabel
- **status** (*str, optional*) – The text of the status tip that appears when the cursor hovers over the label. Defaults to None.
- **tag** (*str, optional*) – Sets the QObject name for this label. Defaults to None.

Returns A label with the desired info.

Return type Qlabel

cyckei.functions.gui.**line_edit**(*label*, *status*, *key*, *connect*)
Creates an editable text edit field with given information

Parameters

- **label** (*str*) – The label in the box when nothing is written there.
- **status** (*str*) – The text of the status tip that appears when the cursor hovers over the label.
- **key** – The parameter that will be needed by the connected function (connect).
- **connect** (*func*) – The function to connect to selecting an item in the dropdown window.

Returns An editable text box object

Return type QLineEdit

cyckei.functions.gui.**message**(*text=None*, *info=None*, *icon=PySide2.QtWidgets.QMessageBox.Icon.Information*, *detail=None*, *confirm=False*)

Show a QMessageBox with given information.

The QMessageBox defaults to simply a popup window, but can also be used to let the user respond with "yes" or "no" and send a corresponding bool. The user can change aspects of the window such as body text, informative text, and detail text.

Parameters

- **text** (*str, optional*) – The body text of the message. Defaults to None.
- **info** (*str, optional*) – In most systems this text is appended to the body text, however in some in appears as smaller text below the body text. Defaults to None.
- **icon** (*int, optional*) – An int (0-4) usable for setting the icon in QMessageBox. Defaults to QMessageBox.Information, an enum from the class representing a 1 int.
- **detail** (*str, optional*) – This is the text that appears in the extra details section. Defaults to None.
- **confirm** (*bool, optional*) – If True gives the user the option to select "Yes" or "No" in the message box. Defaults to False.

Returns returns False if no was selected in the message box, else returns True.

Return type bool

cyckei.functions.gui.**not_none**(*value*)

Sets a None value to "None" string

Parameters **value** (*None*) – Expects a None, but able to handle anything convertabile to a str.

Returns Returns "None" as a string or converts the given value to a string and returns it.

Return type str

cyckei.functions.gui.**style**(*app*, *icon='icon-client.png'*, *highlight='#f05f40'*)

Customizes the style of a QApplication window.

Parameters

- **app** (*QApplication*) – Any object descended from QApplication.
- **icon** (*str, optional*) – The filename, including extension, of an image to be the QIcon for the App. Defaults to "icon-client.png".
- **highlight** (*str, optional*) – The highlight color for the QApp, in HEX color form. Defaults to orange.

cyckei.functions.gui.**text_edit**(*status=None*, *connect=None*, *readonly=False*, *wrap=PySide2.QtWidgets.QPlainTextEdit.LineWrapMode.NoWrap*)

Creates a text box for editing plain text.

Parameters

- **status** (*str, optional*) – The text of the status tip that appears when the cursor hovers over the box. Defaults to None.
- **connect** (*func, optional*) – When text is changed this function will be executed. Defaults to None.
- **readonly** (*bool, optional*) – True makes the text box only readable, not editable. Defaults to False.
- **wrap** (*Const, optional*) – Can be NoWrap or WidgetWidth, depending on if the user wants word wrapping in their text box. Defaults to QPlainTextEdit.NoWrap.

Returns The QPlainTextEdit object.

Return type QPlainTextEdit

7.6 Plugins

Abstract Classes for implementing plugins for Cyckei.

class cyckei.plugins.cyp_base.**BaseController**(*name*, *description*)

Abstract Parent class of plugin controller objects.

Creates default methods for interacting with plugin and handling sources.

description
The description given to the user in the info section.
Type str

logger
The logger for the plugin object. Stored in the Plugins folder, named after the name variable.
Type logging.Logger

name
The name of the plugin object.
Type str

__init__(*name*, *description*)
Inits description, logger, and name. Sets up logging and sources for plugin.
Parameters
- **name** (*str*) – The name of the plugin object.
- **description** (*str*) – The description given to the user in the info section.

cleanup()
Abstract method.
Raises NotImplementedError – Error always raised as this is an abstract method.

get_logger(*name*, *cyckei_plugin_path*)
Connects the plugin to main Cyckei loggers.

Plugin initially tries to connect to to Cyckei's main logging handlers. If this fails, this method establishes a new console handler. Usually this should be as a result of running the plugin independantly.
Parameters
- **name** (*str*) – The name of the plugin object.
- **cyckei_plugin_path** (*str*) – The path to the Plugins folder in the Cyckei folder.

Raises FileNotFoundError – Raised when the path given by cyckei_plugin_path doesn't exist.
Returns The logger for the plugin object. File stored in the Plugins folder, named after the name variable.
Return type logging.Logger

load_sources()
Abstract method. Searches for available sources, and establishes source objects.
Raises NotImplementedError – Error always raised as this is an abstract method.

read(*source*)
Reads data from every source object connected to this plugin controller.

Requires a collection of source objects to be stored in self.sources as a list or dictionary.
Parameters **source** (*int or str*) – The index or key of the source object to be read from. Depends on whether the source objects are stored in a list or dict.
Returns Any type can be returned as this function calls the read function from a source and does no further processing.
Return type Any

class cyckei.plugins.cyp_base.**BaseSource**
Parent class of plugin source object. Controls communication with individual devices or channels.

__init__()
Abstract constructor. No definition

read()
Abstract method. Reads data from the source instrument.
Raises NotImplementedError – Error always raised as this is an abstract method.

cyckei.plugins.cyp_base.**read_all**(*controller*)
Performs the read function on every plugin source stored in the the plugin controller.

Parameters **controller** (BaseController) – A BaseController holding plugin sources to be read from.

Returns A dict with keys as the name of the plugin and values that could be any type. Any type could be returned from reading a plugin source, as there is no type control before this point.

Return type dict

CHAPTER 8

About the Cyckei Project

Cyckei is a battery cycling application designed to carry out charging, discharging, and data collection on lithium-ion cells. It is designed to interface with the Keithley 2602A/B SourceMeter for calorimetry testing, but can be used in a variety or setups.

The application uses a Python-like scripting format in order to write cycles that are carried out on cells. To learn more about scripts, read the *Creating Scripts* section or look at the example below.

```
for i in range(3):
  AdvanceCycle()
  CCCharge(0.1, reports=(("voltage", 0.01), ("time", ":5:")), ends=(("voltage", ">",␣
↪4.2), ("time", ">", "4::")))
  CCDischarge(0.1, reports=(("voltage", 0.01), ("time", ":5:")), ends=(("voltage", "<
↪", 3.0), ("time", ">", "4::")))
  Rest(reports=(("time", "::1"),), ends=(("time", ">", "::15"),))
```

Cyckei is open source, and we encourage users to modify the code to fit a given setup. Details on contributing to the project are in our *Contributing* section.

Cyckei is currently developed and maintained by Vincent Chevrier at Cyclikal LLC, Clark Ohnesorge, and Gabriel Ewig. For more information about Cyclikal, visit cyclikal.com.

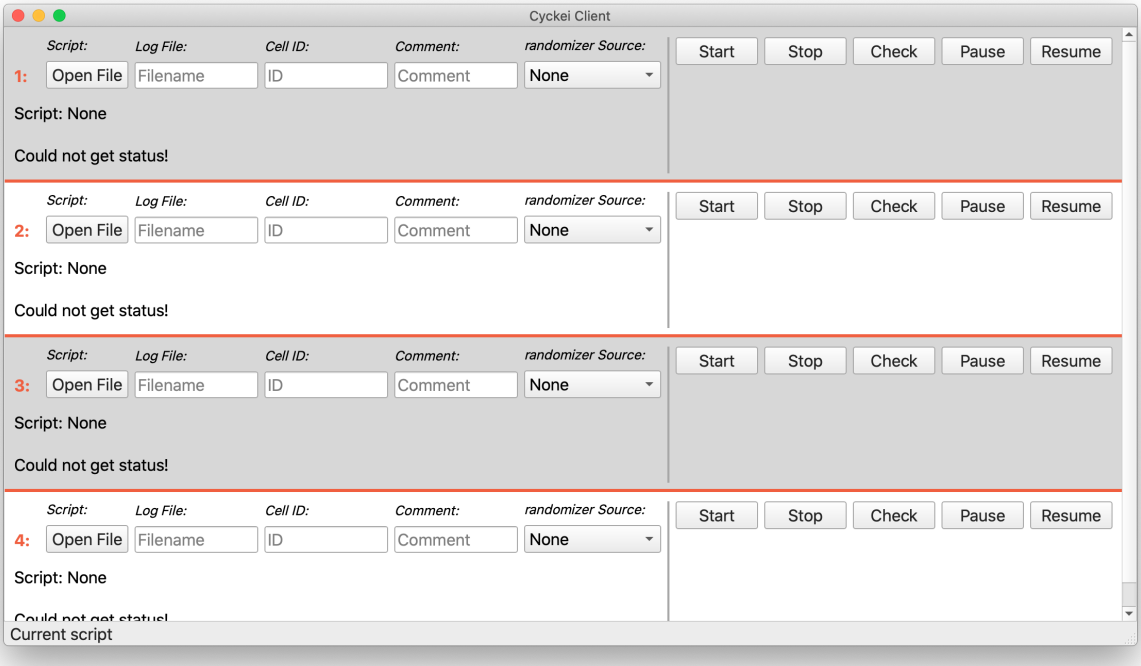


Fig. 1: Screen shot of Cyckei channel tab on Mac OS.

Python Module Index

C

Index

Symbols

A

B

C

D

E

F

G

O

P

Q

R

S

T

U

V

W