TECHNISCHE
UNIVERSITÄT
DRESDEN

Faculty of physics Institute for solid-state and material physics

Python scripts for resistance measurements and temperature control – a short handbook

Julius Müller

February 28, 2022

Contents

1 The `CryoConnectorAPI` module

The `CryoConnectorAPI` module serves as an easy way to control Oxford Cryosystems cryostats through python. Oxford hardware communicates by sending and receiving serial commands, which, on the side of the PC connected to the temperature controller, is handled by the CryoConnector software provided by Oxford. While CryoConnector comes with a GUI to monitor and control connected devices, it also comes with an API to automate the process through the use of scripts. Information about supported devices, as well as status updates and sensor data of any active devices can be read from XML-files found in the working directory of the CryoConnector application. This defaults to %AppData%\CryoConnector\, but can also be configured by the user.

Information on the different ports allowing connections is kept in the `connections.xml` file.

The **`updateConnections()`** method reads this file and notes the indices of any connections that are marked *Active* i.e. the ports that have a cryostat/temperature controller connected to them. When the `CryoConnectorAPI` is initiated, it checks if the provided **`port`** is *Active*. If it is, the index of the port in the file is saved as **`cryo_id`**. If no **`port`** is provided, the user is prompted to select a device from a list of *Active* connections.

The **`updateConnectionInfo()`** method uses the thus chosen **`cryo_id`** to read or update information about the device from the `connections.xml` file.

This file contains the paths to the `status.xml` file as well as to the `[cryostat-name].xml` properties file, which holds lists for things like supported commands and possible alarms. The **`updateProperties()`** method reads the properties file and the **`updateStatus()`** method reads the status file.

The **`get(name)`** method returns the info read from the status file corresponding to the provided `name`, if it exists. It does not parse the `status.xml` file, so `updateStatus()` should be called first.

`CryoConnector` also allows to send commands to connected devices. Any new XML-files written *to* the folder are checked for known commands, and if they contain any, are deleted and the commands forwarded to the temperature controller in the appropriate form. Accordingly, `CryoConnectorAPI` provides the user with the **`command(name, *args)`** method. When `command(name, *args)` is called, it first checks whether the variable `name` is in the list of commands understood by the connected device, and whether the number of arguments passed coincides with the number given in the list. It also checks if maximum or minimum values are defined for the device and adjusts the setpoint if necessary. It then creates an XML-tree containing the command and writes it to a new file in the CryoConnector working directory.

2 The `Cryostat` module

The `Cryostat.py` module aims to unify communication with temperature controllers from different manufacturers in order to make compatibility of the `ResistanceMeasurement` module (see chapter 3) more easily expandable. It currently supports:

- Oxford Cryosystems N-HeliX
- Lake Shore Cryotronics Model 336

The module has been built with the N-HeliX temperature controller in mind, but all hardware that uses the `CryoConnector` software to communicate with a connected PC should in principle be supported. Likewise, other Lake Shore devices should also be compatible, as long as they have at least two inputs (A and B), two outputs (1 and 2) and support PID zone control mode.

`CryoConnector` is necessary for operating Oxford hardware and if it is not found on the PC, the user will be offered to install it before continuing. For Lake Shore hardware, the official python driver is used to implement the same functionality.

Connect to a device

When the `Cryostat(device=”, IP=”, port=”, CCWorkingFolder='default', testMode=0)` class is instantiated, it checks if a supported **`device`** type has been passed to it, along with either a **`port`** number or an **`IP`** address. It attempts to connect to the device with the provided `port` or `IP`. If no device type and neither `port` nor `IP` are provided, the user is prompted with a small menu to choose the device and input either the corresponding `IP` or, for the case of an Oxford cooler, choose their temperature controller from a list as described in chapter 1.

Read status information and temperature

The **`updateStatus()`** method of `Cryostat` is based on the `CryoConnectorAPI` method of the same name (see chapter 1) and saves/updates the most important operational parameters of the device as class properties. These properties are:

`Cryostat.sampleTemp`	– Sample temperature in K
`Cryostat.cryoTemp`	– Cryostat temperature in K
`Cryostat.deviceName`	– Name or handle of the device
`Cryostat.deviceStatus`	– Info about the status of the device, e.g. 'Running'
`Cryostat.phaseStatus`	– Info about the current phase, e.g. 'Hold at 100K'
`Cryostat.alarmStatus`	– Info about any alarm that occurs, e.g. 'Sensor fail' etc.
`Cryostat.alarmLevel`	– Numeric indicating the gravity of the alarm, from 0 to 4

This means that the above readings, including the **temperature**, are only as recent as the last time `updateStatus()` has been called.

In the case of Lake Shore temperature controllers, a number of functions are called and operations performed in order to replicate the same parameters.

Read or set calibration curves

While the N-HeliX provides a sample temperature, the temperature reading of the Model 336 depends on the sensor placement, which might be some ways off the sample position. For this reason, the `sampleTemp` variable is translated from the `cryoTemp` variable using the `TA_to_Tsample(temperature_A)` method via a **calibration curve**. The reverse function `Tsample_to_TA(temperature_sample)` also exists for translating desired sample temperatures to sensor temperature setpoints.

The calibration file paths can be read using `getCalibrationCurves()`; they can be changed using `setCalibrationCurves(TA_to_Tsample_name, Tsample_to_TA_name)`. Calibration curve files need to be located in the `./calibration_curves/` directory.

Control temperature controller

The connected cooler is controlled with the following commands. Not all commands are supported by all coolers, `commands()` returns a list of commands understood by the chosen device.

`Cryostat.ramp(rampRate, temp)`	– Cryostat changes to new sample temperature at a controlled rate.
`Cryostat.plat(duration)`	– Cryostat holds current temperature for a period specified in minutes.
`Cryostat.hold()`	– Cryostat holds current temperature indefinitely.
`Cryostat.cool(temp)`	– Cryostat changes to new temperature as quickly as possible.
`Cryostat.purge()`	– Cryostat is stopped and cooler is warmed to room temperature.
`Cryostat.suspend()`	– Cryostat enters temporary hold.
`Cryostat.resume()`	– Cryostat exits temporary hold.
`Cryostat.end(rampRate)`	– Cryostat ramps to 300K at a specified rate and then shuts down.
`Cryostat.restart()`	– Restart cryostat after shut down.
`Cryostat.stop()`	– Stop operation immediately.

Currently, only the commands `ramp(rampRate, temp)` and `stop()` have been implemented for the Lake Shore Model 336 temperature controller, as it doesn't natively support the other commands the same way Oxford coolers do. However, they could very easily be realized using variations on the `ramp` command.

There is another way in which the Lake Shore control differs from the Oxford cooler control as implemented in this module. While in the case of an Oxford Cryosystems temperature controller, the `ramp(rampRate, temp)` function simply calls `command(name='Ramp', rampRate, temp)` (see chapter 1), the behavior is somewhat different for a Lake Shore device (such as the Model 336) which are assumed to have two heaters and two temperature sensors. Here, it tells heater 1 to ramp to `temp` at the specified `rampRate` and simultaneously sets heater 2 to ramp to a slightly lower temperature setpoint using pre-configured zone settings. In this way, heater 2 attempts to offset somewhat the negative heat-input of the cold head, enabling heater 1 to accurately control the temperature at input A using the same P, I, and D settings for the entire temperature range of the cryostat. Internally, `Cryostat.py` uses `Cryostat.zoneRangeTable` to configure PID zone settings and send them to the temperature controller.

Expand the module for additional cooling hardware

Adding more temperature controllers to the module should be relatively straight-forward. If you wish to do so, work your way through the **`__init__`** source code (which features documentation for this very purpose):

The new hardware has to be added to **`Cryostat.supported_manufacturers`**, **`Cryostat.supported_devices`** and **`Cryostat.supported_commands`**. The position of the device name in `Cryostat.supported_devices` determines the **`Cryostat.index`** variable. This variable is used throughout the module to decide which code to perform specifically for the connected cooler.

The next step (still in `__init__`) is to write code to define how to connect to and initialize the new cooler. This might also mean adjusting the GUI functions **`chooseCryo()`** and **`_okButton()`**. Specifically the latter needs to be expanded if the device to be added is from a manufacturer not yet in the list of supported manufacturers.

After that, simply add (**`if self.index == x:`**)–blocks (x is the index for your cooler) with the appropriate code to the **`updateStatus()`**-method as well as to all the **commands** the device should support.

3 The `ResistanceMeasurement` module

`ResistanceMeasurement` is intended as a GUI-based tool to control, first and foremost, a Keithley sourcemeter in conjunction with a Keithley voltmeter, to set up and perform resistance measurements using a four point contact geometry. The user is able to manually adjust the current supplied to the connected sample for measurements, or they can run an automatic calibration routine, which ramps up the current, until the measurement spread falls within a desired window of error. They can then run continuous resistance measurements to track the evolution of resistance over time, or take isolated measurements at temperature points that the user needs to input individually.

Optionally, supported temperature controllers can also be connected in addition to the source- and voltmeter. In this case the user is given the option to configure entire measurement runs wherein the software successively and automatically ramps to desired temperatures, measuring the resistance at certain user-defined intervals along the way. When using the Lake Shore Model 336, the Lake Shore Chart Recorder can be used simultaneously for easy monitoring of the heaters and temperature sensors.

Connect to measurement hardware

When the program is started up, the user is presented with a small menu to connect to the experimental hardware.

Keithley sourcemeter and voltmeter should be connected to the PC via USB or similar, and will be initiated using the National Instruments VISA API internally. The user only needs to select the appropriate port numbers. These can be configured within the built-in menus of the Keithley devices; the default ports are already pre-selected in the pop-up menu of the ResistanceMeasurement module.

The program will work with just the Keithley devices connected, but obviously, no fully automatic temperature runs are available in this case. Instead, the temperature parameter can be changed manually in the settings menu (see section Configure settings below).

Connecting a temperature controller initiates the `Cryostat` module as described in chapter The `Cryostat` module above.

When all required hardware has been set up, the experiment can be performed from the main GUI window. The appearance of the program in action is examplarily depicted in fig. 3.1, the different functions will briefly be introduced in the following sections.

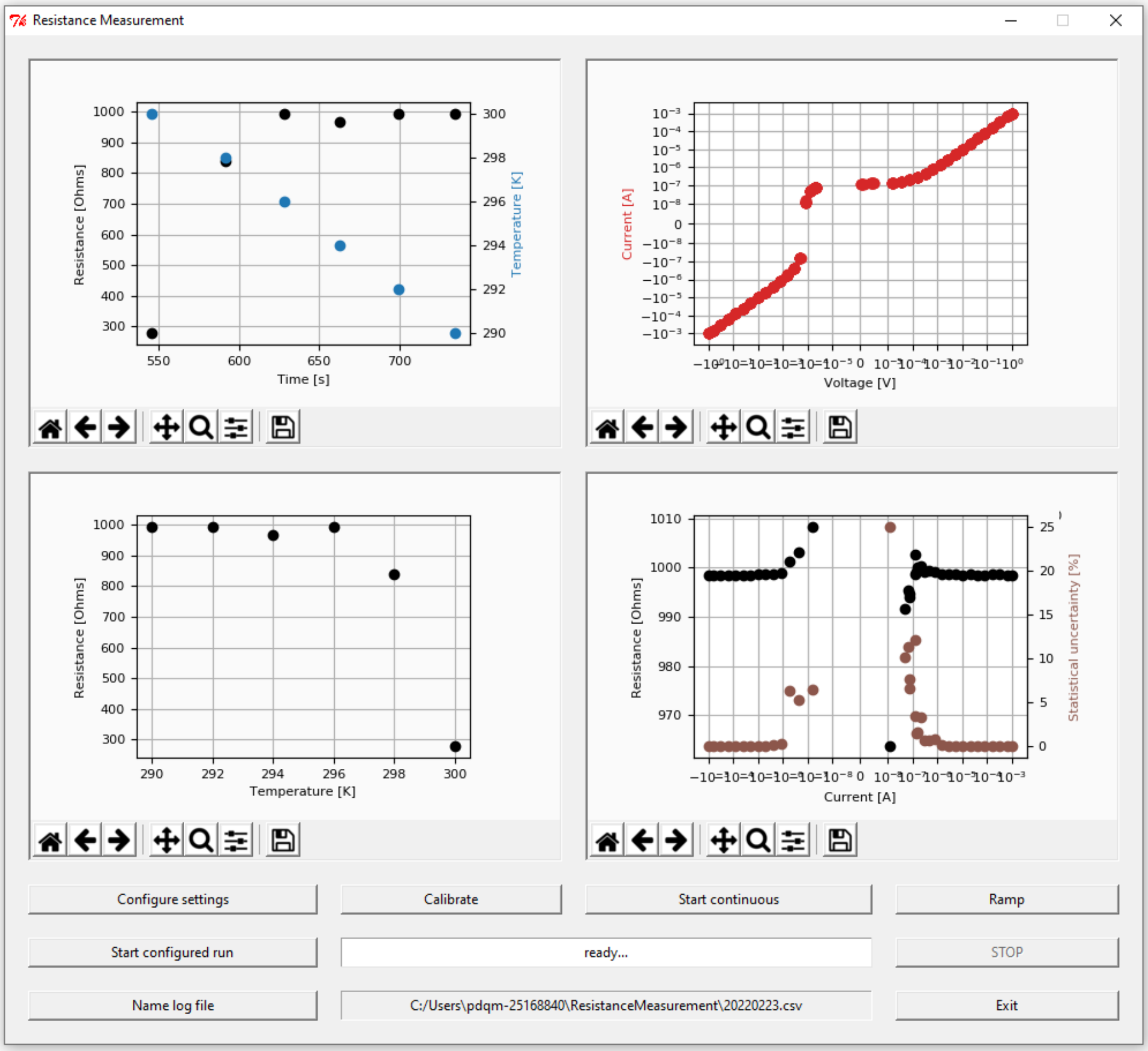

Figure 3.1: The main Window with graphs for active measurements on the left and calibration on the right. All measurement functions are easily accessible from the controls area at the bottom.

Configure settings

Opens a sub-menu from which a number of essential parameters can be adjusted, including the **current setpoint**, the temperature sensor **calibration curves** if using a cryo cooler, or the **temperature** parameter if not using a cryo cooler, and more.

Also here can be found the option to configure an **automated temperature run**: These consist of individual stages, each of which can be set up with a target temperature to ramp to, an associated ramp rate, the step width for intermediate resistance probing, and a wait time to be observed before each resistance measurement (if you know that the temperature needs some time to stabilize).

Calibrate

Starts the calibration routine, which ramps up the voltage from `U_min` to `U_max` (configurable in the settings menu), alternating between positive and negative current directions, and records the **IV-curve**. Several data-points are collected at each voltage setpoint. The top right figure shows the IV-curve on a double-symmetric-logarithmic scale in real time, the figure below tracks the resistance, as well as the statistical fluctuations in resistance measurements as a function of the applied current. When `U_max` is reached, or when the user stops the calibration, an "optimal" current setpoint is calculated, based on the following merit function:

$$f \quad = \quad A \cdot \frac{\left\| \frac{d^2}{dU^2} I \right\|}{max\left(\left\| \frac{d^2}{dU^2} I \right\| \right)} \quad + \quad B \cdot \frac{\Delta R}{max\left(\Delta R\right)} \quad + \quad C \cdot \frac{U^2}{max\left(U^2\right)} \tag{3.1}$$

This merit function attempts to balance three different influencing factors of varying current/voltage setpoints. The first is the second derivative of the IV-curve, i.e. how linear, or 'ohmic', the sample behaves in that region. The second factor is the error in R, i.e. the precision of repeated measurements, and the last is the voltage squared, which represents undesirable ohmic heating power. The weighting factors A, B and C may be adjusted in the settings menu.

A pop-up message informs of the calculated current setpoint, but can be overwritten. The current setpoint may be changed at any time in the settings menu.

Start continuous

Starts continuously probing and plotting the resistance of the connected sample. Both the top left and bottom left figures are updated in real time. The same button also stops the measurement.

Ramp

Opens a small sub-menu, where the user can manually set the cooler to ramp to a specified temperature at a specified ramp rate. The button is inactive when no cooler is connected.

Start configured run / Take measurement

Starts the **automated temperature run** configured in the settings menu. Both the top left and bottom left plots are updated in real time.

When no cooler is connected, the function is replaced by a single measurement button, which quickly probes the resistance several times and calculates an average resistance. Updates top left and bottom left plots. This function in conjunction with the option the change the temperature parameter in the settings menu enables **semi-manual temperature runs** when using some means of temperature control that is not compatible with this python solution.

Stop

Stops the **automated temperature run**.

Name log file

Allows to rename the **log file**. Calibration, continuous measurement and automatic measurement **overwrite** the log file (the user is warned if the file already exists), the single-shot measurement button **appends** one line to the log file.

Exit

This button, after asking confirmation, stops all measurement activity and closes the program.

Bibliography

[1] Oxford Cryosystems. *Oxford Cryosystems serial line communication protocols*. 2020. URL: `https://connect.oxcryo.com/serialcomms/index.html`.

[2] Oxford Cryosystems. *CryoConnector Documentation*. 2020. URL: `https://sites.google.com/a/oxcryo.com/cryoconnector/home`.

[3] Lake Shore Cryotronics. *Lake Shore Python Driver*. 2021. URL: `https://pypi.org/project/lakeshore/`.

[4] Keithley Instruments. *Model 2182/2182A Nanovoltmeter User's Manual*. May 2017. URL: `https://download.tek.com/manual/2182A-900-01_May_2017.pdf`.

[5] Keithley Instruments. *Series 2400 SourceMeter User's Manual*. Sept. 2011. URL: `https://download.tek.com/manual/2400S-900-01_K-Sep2011_User.pdf`.