keithley2600 Documentation

Release 2.0.2

Sam Schott

Sep 23, 2021

CONTENTS

This documentation provides both a quick introduction to keithley2600 and a detailed reference of the API:

CHAPTER

ONE

KEITHLEY2600

A full Python driver for the Keithley 2600B series of source measurement units. An accompanying GUI is provided by the sister project keithleygui. Documentation is available at https://keithley2600.readthedocs.io.

1.1 About

This driver provides access to base commands and higher level functions such as IV measurements, transfer and output curves, etc. Base commands replicate the functionality and syntax from the Keithley's internal TSP Lua functions. This is possible because the Lua programming language has a very limited syntax which can be represented by a subset of Python syntax.

All Keithley commands are dynamically queried from the Keithley itself after a successful connection. This means that essentially all Keithley instruments which use TSP scripting are supported and any commands introduced in the future will be recognised automatically (barring changes to the Lua syntax itself). Please refer to the respective reference manuals for a list of commands available on a particular model, for instance the Keithley 2600B reference manual.

This dynamic approach however means that most attributes will only be defined after connecting to an instrument. Several higher level functions for current-voltage sweeps are defined by the driver itself and may use functionality which is not available on some models. They have been tested with a Keithley 2612B.

Warning: There are currently no checks for allowed arguments by the driver itself. Passing invalid arguments to a Keithley command will fail silently in Python but will display an error message on the Keithley itself. To enable command checking, set the keyword argument `raise_keithley_errors = True` in the constructor. When set, most Keithley errors will be raised as Python errors. This is done by reading the Keithley's error queue after every command and will therefore result in some communication overhead.

1.2 Installation

Install the stable version from PyPi by running

```
$ pip install keithley2600
```

or the latest development version from GitHub:

```
$ pip install git+https://github.com/OE-FET/keithley2600
```

1.3 Usage

Connect to the Keithley 2600 and perform some base commands:

```
from keithley2600 import Keithley2600

k = Keithley2600('TCPIP0::192.168.2.121::INSTR')

k.smua.source.output = k.smua.OUTPUT_ON   # turn on SMUA
k.smua.source.levelv = -40  # sets SMUA source level to -40V
v = k.smua.measure.v()  # measures and returns the SMUA voltage
i = k.smua.measure.i()  # measures current at smuA

k.smua.measure.v(k.smua.nvbuffer1)  # measures the voltage, stores the result in␣
↪buffer
k.smua.nvbuffer1.clear()  # clears nvbuffer1 of SMUA
```

Higher level commands defined in the driver:

```
data = k.read_buffer(k.smua.nvbuffer1)  # reads all entries from nvbuffer1 of SMUA
errs = k.read_error_queue()  # gets all entries from error queue

k.set_integration_time(k.smua, 0.001)  # sets integration time in sec
k.apply_voltage(k.smua, 10)  # turns on and applies 10V to SMUA
k.apply_current(k.smub, 0.1)  # sources 0.1A from SMUB
k.ramp_to_voltage(k.smua, 10, delay=0.1, stepSize=1)  # ramps SMUA to 10V in steps of␣
↪1V

# sweep commands
k.voltage_sweep_single_smu(
    k.smua, range(0, 61), t_int=0.1, delay=-1, pulsed=False
)
k.voltage_sweep_dual_smu(
    smu1=k.smua,
    smu2=k.smub,
    smu1_sweeplist=range(0, 61),
    smu2_sweeplist=range(0, 61),
    t_int=0.1,
    delay=-1,
    pulsed=False,
)
k.transfer_measurement( ... )
k.output_measurement( ... )
```

Singleton behaviour:

Once a Keithley2600 instance with a visa address `'address'` has been created, repeated calls to `Keithley2600('address')` will return the existing instance instead of creating a new one. This prevents the user from opening multiple connections to the same instrument simultaneously and allows easy access to a Keithley2600 instance from different parts of a program. For example:

```
>>> from keithley2600 import Keithley2600
>>> k1 = Keithley2600('TCPIP0::192.168.2.121::INSTR')
>>> k2 = Keithley2600('TCPIP0::192.168.2.121::INSTR')
>>> print(k1 is k2)
True
```

Data structures:

The methods `voltage_sweep_single_smu` and `voltage_sweep_dual_smu` return lists with the measured voltages and currents. The higher level commands `transfer_measurement` and `output_measurement` return `ResultTable` objects which are somewhat similar to pandas dataframes but include support for column units. `ResultTable` stores the measurement data internally as a numpy array and provides information about column titles and units. It also provides a dictionary-like interface to access columns by name, methods to load and save the data to text files, and live plotting of the data (requires matplotlib).

For example:

```
import time
from  keithley2600 import Keithley2600, ResultTable

k = Keithley2600('TCPIP0::192.168.2.121::INSTR')

# create ResultTable with two columns
rt = ResultTable(
    column_titles=['Voltage', 'Current'],
    units=['V', 'A'],
    params={'recorded': time.asctime(), 'sweep_type': 'iv'},
)

# create live plot which updates as data is added
rt.plot(live=True)

# measure some currents
for v in range(0, 20):
    k.apply_voltage(k.smua, 10)
    i = k.smua.measure.i()
    rt.append_row([v, i])

# save the data
rt.save('~/iv_curve.txt')
```

See the documentation for all available methods.

1.4 Backend selection

keithley2600 uses PyVISA to connect to instruments. PyVISA supports both proprietray IVI libraries such as NI-VISA, Keysight VISA, R&S VISA, tekVISA etc. and the purely Python backend PyVISA-py. You can select a specific backend by giving its path to the `Keithley2600` constructor in the `visa_library` argument. For example:

```
from  keithley2600 import Keithley2600

k = Keithley2600('TCPIP0::192.168.2.121::INSTR', visa_library='/usr/lib/libvisa.so.7')
```

keithley2600 defaults to using the PyVISA-py backend, selected by `visa_library='@py'`, since this is only a pip-install away. If you pass an empty string, keithley2600 will use an installed IVI library if it can find one in standard locations or fall back to PyVISA-py otherwise.

You can find more information about selecting the backend in the PyVISA docs.

1.5 System requirements

- Python 3.6 or higher

1.6 Documentation

- API documentation of keithley2600: https://keithley2600.readthedocs.io/en/latest/
- Keithley 2600 reference manual with all commands: https://www.tek.com/keithley-source-measure-units/smu-2600b-series-sourcemeter-manual-8

CHAPTER

TWO

MODULES

This section gives an overview of keithley2600's modules:

2.1 Keithley driver

Core driver with the low level functions.

exception keithley_driver.**KeithleyIOError**
Bases: Exception

Raised when no Keithley instrument is connected.

exception keithley_driver.**KeithleyError**
Bases: Exception

Raised for error messages from the Keithley itself.

class keithley_driver.**Keithley2600Base**(*visa_address: str*, *visa_library: str = '@py'*, *raise_keithley_errors: bool = False*, ***kwargs*)
Bases: keithley_driver.KeithleyClass

Keithley2600 driver

Keithley driver for base functionality. It replicates the functionality and syntax from the Keithley TSP commands, as provided by the Lua scripting language. Attributes are created on-demand if they correspond to Keithley TSP commands.

Parameters

- **visa_address** – Visa address of the instrument.
- **visa_library** – Path to visa library. Defaults to "@py" for pyvisa-py but another IVI library may be appropriate (NI-VISA, Keysight VISA, R&S VISA, tekVISA etc.). If an empty string is given, an IVI library will be used if installed and pyvisa-py otherwise.
- **raise_keithley_errors** – If `True`, all Keithley errors will be raised as Python errors instead of being ignored. This causes significant communication overhead because the Keithley's error queue is read after each command. Defaults to `False`.
- **kwargs** – Keyword arguments passed on to the visa connection, for instance baude-rate or timeout. If not given, reasonable defaults will be used.

Variables

- **connection** – Attribute holding a reference to the actual connection.
- **connected** – `True` if connected to an instrument, `False` otherwise.
- ***busy*** – `True` if a measurement is running, `False` otherwise.

- **CHUNK_SIZE** – Maximum length of lists which can be sent to the Keithley. Longer lists will be transferred in chunks.

Note: See the Keithley 2600 reference manual for all available commands and arguments. A dictionary of available commands will be loaded on access from the Keithley at runtime, if connected.

Examples

```
>>> keithley = Keithley2600Base('TCPIP0::192.168.2.121::INSTR')
>>> keithley.smua.measure.v()  # measures voltage at smuA
>>> keithley.smua.source.levelv = -40  # applies -40V to smuA
```

connect(***kwargs*) → bool
Connects to Keithley.

Parameters **kwargs** – Keyword arguments for Visa connection.

Returns Whether the connection succeeded.

disconnect() → None
Disconnects from Keithley.

class keithley_driver.**Keithley2600**(*visa_address: str*, *visa_library: str = '@py'*, *raise_keithley_errors: bool = False*, ***kwargs*)
Bases: *keithley_driver.Keithley2600Base*

Keithley2600 driver with high level functionality

Keithley driver with access to base functions and higher level functions such as IV measurements, transfer and output curves, etc. Inherits from *Keithley2600Base*. Base commands replicate the functionality and syntax of Keithley TSP functions.

Parameters

- **visa_address** – Visa address of the instrument.
- **visa_library** – Path to visa library. Defaults to "@py" for pyvisa-py but another IVI library may be appropriate (NI-VISA, Keysight VISA, R&S VISA, tekVISA etc.). If an empty string is given, an IVI library will be used if installed and pyvisa-py otherwise.
- **raise_keithley_errors** – If `True`, all Keithley errors will be raised as Python errors instead of being ignored. This causes significant communication overhead because the Keithley's error queue is read after each command. Defaults to `False`.
- **kwargs** – Keyword arguments passed on to the visa connection, for instance baude-rate or timeout. If not given, reasonable defaults will be used.

Variables

- **connection** – Attribute holding a reference to the actual connection.
- **connected** – `True` if connected to an instrument, `False` otherwise.
- ***busy*** (*bool*) – `True` if a measurement is running, `False` otherwise.

Examples *Base commands from Keithley TSP*:

```
>>> k = Keithley2600('TCPIP0::192.168.2.121::INSTR')
>>> volts = k.smua.measure.v()  # measures and returns the smuA␣
↪voltage
```

(continues on next page)

(continued from previous page)

```
>>> k.smua.source.levelv = -40  # sets source level of smuA
>>> k.smua.nvbuffer1.clear()  # clears nvbuffer1 of smuA
```

New mid-level commands:

```
>>> data = k.read_buffer(k.smua.nvbuffer1)
>>> errs = k.read_error_queue()
>>> k.set_integration_time(k.smua, 0.001) # in sec
```

```
>>> k.apply_voltage(k.smua, -60) # applies -60V to smuA
>>> k.apply_current(k.smub, 0.1) # sources 0.1A from smuB
>>> k.ramp_to_voltage(k.smua, 10, delay=0.1, step_size=1)
```

```
>>> # voltage sweeps, single and dual SMU
>>> k.voltage_sweep_single_smu(smu=k.smua, smu_sweeplist=range(0, 61),
...                            t_int=0.1, delay=-1, pulsed=False)
>>> k.voltage_sweep_dual_smu(smu1=k.smua, smu2=k.smub,
...                          smu1_sweeplist=range(0, 61),
...                          smu2_sweeplist=range(0, 61),
...                          t_int=0.1, delay=-1, pulsed=False)
```

New high-level commands:

```
>>> data1 = k.output_measurement(...) # records output curve
>>> data2 = k.transfer_measurement(...) # records transfer curve
```

apply_current(*smu: keithley_driver.KeithleyClass*, *curr: float*) → None
Turns on the specified SMU and sources a current.

Parameters

- **smu** – A keithley smu instance.
- **curr** – Current to apply in Ampere.

apply_voltage(*smu: keithley_driver.KeithleyClass*, *voltage: float*) → None
Turns on the specified SMU and applies a voltage.

Parameters

- **smu** – A keithley smu instance.
- **voltage** – Voltage to apply in Volts.

property busy
True if a measurement is running, False otherwise.

measure_current(*smu: keithley_driver.KeithleyClass*) → float
Measures a current at the specified SMU.

Parameters **smu** – A keithley smu instance.

Returns Measured current in Ampere.

measure_voltage(*smu: keithley_driver.KeithleyClass*) → float
Measures a voltage at the specified SMU.

Parameters **smu** – A keithley smu instance.

Returns Measured voltage in Volts.

output_measurement(*smu_gate: keithley_driver.KeithleyClass, smu_drain: keithley_driver.KeithleyClass, vd_start: float, vd_stop: float, vd_step: float, vg_list: Sequence[float], t_int: float, delay: float, pulsed: bool*) → keithley2600.result_table.FETResultTable

Records an output curve with forward and reverse sweeps and returns the results in a `sweep_data.TransistorSweepData` instance.

Parameters

- **smu_gate** – Keithley smu attached to gate electrode.
- **smu_drain** – Keithley smu attached to drain electrode.
- **vd_start** (*float*) – Start voltage of output sweep in Volt.
- **vd_stop** (*float*) – End voltage of output sweep in Volt.
- **vd_step** (*float*) – Voltage step size for output sweep in Volt.
- **vg_list** – List of gate voltage steps in Volt. Can be a numpy array, list, tuple, range / xrange.
- **t_int** (*float*) – Integration time per data point. Must be between 0.001 to 25 times the power line frequency (50Hz or 60Hz).
- **delay** (*float*) – Settling delay before each measurement. A value of -1 automatically starts a measurement once the current is stable.
- **pulsed** (*bool*) – Select pulsed or continuous sweep. In a pulsed sweep, the voltage is always reset to zero between data points.

Returns Output curve data.

play_chord(*notes: Tuple[str, ...] = ('C6', 'E6', 'G6')*, *durations: Union[float, Iterable[float]] = 0.3*) → None

Plays a chord on the Keithley.

Parameters

- **notes** – List of notes in scientific pitch notation, for instance `['F4', 'Ab4', 'C4']` for a f-minor chord in the 4th octave. Defaults to c-major in the 6th octave.
- **durations** – List of durations for each note in sec. If a single float is given, all notes will have the same duration. Defaults to 0.3 sec.

ramp_to_voltage(*smu: keithley_driver.KeithleyClass*, *target_volt: float*, *delay: float = 0.1*, *step_size: float = 1*) → None

Ramps up the voltage of the specified SMU.

Parameters

- **smu** – A keithley smu instance.
- **target_volt** – Target voltage in Volts.
- **step_size** – Size of the voltage steps in Volts.
- **delay** – Delay between steps in sec.

static read_buffer(*buffer: keithley_driver.KeithleyClass*) → List[float]

Reads buffer values and returns them as a list. This can be done more quickly by calling `buffer.readings` but such a call may fail due to I/O limitations of the keithley if the returned list is too long.

Parameters **buffer** – A keithley buffer instance.

Returns A list with buffer readings.

read_error_queue() → List[Tuple[Union[float, str, bool, None, keithley_driver._LuaFunction, keithley_driver._LuaTable], ...]]

Returns all entries from the Keithley error queue and clears the queue.

Returns List of errors from the Keithley error queue. Each entry is a tuple `(error_code, message, severity, error_node)`. If the queue is empty, an empty list is returned.

send_trigger() → None

Manually sends a trigger signal to the Keithley. This can be used for instance to start a pre-programmed sweep.

set_integration_time(*smu: keithley_driver.KeithleyClass*, *t_int: float*) → None

Sets the integration time of SMU for measurements in sec.

Parameters

- **smu** – A keithley smu instance.
- **t_int** – Integration time in sec. Value must be between 1/1000 and 25 power line cycles (50Hz or 60 Hz).

Raises `ValueError` for too short or too long integration times.

transfer_measurement(*smu_gate: keithley_driver.KeithleyClass, smu_drain: keithley_driver.KeithleyClass, vg_start: float, vg_stop: float, vg_step: float, vd_list: Sequence[float], t_int: float, delay: float, pulsed: bool*) → keithley2600.result_table.FETResultTable

Records a transfer curve with forward and reverse sweeps and returns the results in a `sweep_data.TransistorSweepData` instance.

Parameters

- **smu_gate** – Keithley smu attached to gate electrode.
- **smu_drain** – Keithley smu attached to drain electrode.
- **vg_start** – Start voltage of transfer sweep in Volt.
- **vg_stop** – End voltage of transfer sweep in Volt.
- **vg_step** – Voltage step size for transfer sweep in Volt.
- **vd_list** – List of drain voltage steps in Volt. Can be a numpy array, list, tuple, range / xrange. Optionally, you can also pass the string `"trailing"` for the drain voltage to always follow the gate voltage. This ensures that the FET is always at the edge of "saturation".
- **t_int** – Integration time per data point. Must be between 0.001 to 25 times the power line frequency (50Hz or 60Hz).
- **delay** – Settling delay before each measurement. A value of -1 automatically starts a measurement once the current is stable.
- **pulsed** (*bool*) – Select pulsed or continuous sweep. In a pulsed sweep, the voltage is always reset to zero between data points.

Returns Transfer curve data.

voltage_sweep_dual_smu(*smu1: keithley_driver.KeithleyClass, smu2: keithley_driver.KeithleyClass, smu1_sweeplist: Sequence[float], smu2_sweeplist: Sequence[float], t_int: float, delay: float, pulsed: bool*) → Tuple[List[float], List[float], List[float], List[float]]

Sweeps voltages at two SMUs. Measures and returns current and voltage during sweep.

Parameters

- **smu1** – 1st keithley smu instance to be swept.
- **smu2** – 2nd keithley smu instance to be swept.
- **smu1_sweeplist** – Voltages to sweep at `smu1` (can be a numpy array, list, tuple or any other iterable with numbers).
- **smu2_sweeplist** – Voltages to sweep at `smu2` (can be a numpy array, list, tuple or any other iterable with numbers).
- **t_int** – Integration time per data point. Must be between 0.001 to 25 times the power line frequency (50Hz or 60Hz).
- **delay** – Settling delay before each measurement. A value of -1 automatically starts a measurement once the current is stable.
- **pulsed** – Select pulsed or continuous sweep. In a pulsed sweep, the voltage is always reset to zero between data points.

Returns Lists of voltages and currents measured during the sweep (in Volt and Ampere, respectively): `(v_smu1, i_smu1, v_smu2, i_smu2)`.

voltage_sweep_single_smu(*smu: keithley_driver.KeithleyClass, smu_sweeplist: Sequence[float], t_int: float, delay: float, pulsed: bool*) → Tuple[List[float], List[float]]

Sweeps the voltage through the specified list of steps at the given SMU. Measures and returns the current and voltage during the sweep.

Parameters

- **smu** – A keithley smu instance.
- **smu_sweeplist** – Voltages to sweep through (can be a numpy array, list, tuple or any other iterable of numbers).
- **t_int** – Integration time per data point. Must be between 0.001 to 25 times the power line frequency (50Hz or 60Hz).
- **delay** – Settling delay before each measurement. A value of -1 automatically starts a measurement once the current is stable.
- **pulsed** – Select pulsed or continuous sweep. In a pulsed sweep, the voltage is always reset to zero between data points.

Returns Lists of voltages and currents measured during the sweep (in Volt and Ampere, respectively): `(v_smu, i_smu)`.

2.2 Classes to hold sweep results

Submodule defining classes to store, plot, and save measurement results.

class result_table.**ColumnTitle**(*name: str*, *unit: Optional[str] = None*, *unit_fmt: str = '[{}]'*)

Bases: `object`

Class to hold a column title.

Parameters

- **name** – Column name.
- **unit** – Column unit.

- **unit_fmt** – Formatting directive for units when generating string representations. By default, units are enclosed in square brackets (e.g., "Gate voltage [V]").

class result_table.**FETResultTable**(*column_titles: Optional[List[str]] = None*, *units: Optional[List[str]] = None*, *data: Optional[Sequence[float]] = None*, *params: Optional[Dict[str, Any]] = None*)

Bases: *result_table.ResultTable*

Class to handle, store and load transfer and output characteristic data of FETs. `TransistorSweepData` inherits from *ResultTable* and overrides the plot method.

plot(**args*, ***kwargs*) → result_table.ResultTablePlot

Plots the transfer or output curves. Overrides *ResultTable.plot()*. Absolute values are plotted, on a linear scale for output characteristics and a logarithmic scale for transfer characteristics. Takes the same arguments as *ResultTable.plot()*.

Returns *ResultTablePlot* instance with Matplotlib figure.

Raises **ImportError** – If import of matplotlib fails.

class result_table.**ResultTable**(*column_titles: Optional[List[str]] = None*, *units: Optional[List[str]] = None*, *data: Optional[Sequence[float]] = None*, *params: Optional[Dict[str, Any]] = None*)

Bases: `object`

Class that holds measurement data. All data is stored internally as a numpy array with the first index designating rows and the second index designating columns.

Columns must have titles and can have units. It is possible to access the data in a column by its title in a dictionary type notation.

Parameters

- **column_titles** (*list*) – List of column titles.
- **units** (*list*) – List of column units.
- **data** – Numpy array holding the data with the first index designating rows and the second index designating columns. If `data` is `None`, an empty array with the required number of columns is created.
- **params** – Dictionary of measurement parameters.

Examples Create a *ResultTable* to hold current-vs-time data:

```
>>> import time
>>> import numpy as np
>>> from  keithley2600 import ResultTable
>>> # create dictionary of relevant measurement parameters
>>> pars = {'recorded': time.asctime(), 'sweep_type': 'iv'}
>>> # create ResultTable with two columns
>>> rt = ResultTable(['Voltage', 'Current'], ['V', 'A'], params=pars)
>>> # create a live plot of the data
>>> fig = rt.plot(live=True)
```

Create a `Keithley2600` instance and record some data:

```
>>> from  keithley2600 import Keithley2600
>>> k = Keithley2600('TCPIP0::192.168.2.121::INSTR')
>>> for v in range(11):  # measure IV characteristics from 0 to 10 V
...     k.apply_voltage(k.smua, 10)
...     i = k.smua.measure.i()
```

(continues on next page)

(continued from previous page)

```
...     rt.append_row([v, i])
...     time.sleep(1)
```

Print a preview of data to the console:

```
>>> print(rt)
Voltage [V]   Current [A]
0.0000e+00   1.0232e-04
1.0000e+00   2.2147e-04
2.0000e+00   3.6077e-04
3.0000e+00   5.2074e-04
4.0000e+00   6.9927e-04
```

Save the recorded data to a tab-delimited text file:

```
>>> rt.save('~/Desktop/stress_test.txt')
```

append_column(*data: Sequence[float], name: str, unit: Optional[str] = None*) → None
Appends a single column to the data array.

Parameters

- **data** – Sequence with the same number of elements as rows in the data array.
- **name** – Name of new column.
- **unit** – Unit of values in new column.

append_columns(*data: Sequence[float], column_titles: List[str], units: Optional[List[str]] = None*) → None
Appends multiple columns to data array.

Parameters

- **data** – List of columns to append.
- **column_titles** – List of column titles (strings).
- **units** – List of units for new columns (strings).

append_row(*data: Sequence[float]*) → None
Appends a single row to the data array.

Parameters **data** – Sequence with the same number of elements as columns in the data array.

append_rows(*data: Sequence[float]*) → None
Appends multiple rows to the data array.

Parameters **data** – List of lists or numpy array with dimensions matching the data array.

clear_data() → None
Clears all data.

property column_names
List of strings with column names.

property column_units
List of strings with column units.

get_column(*i: int*) → numpy.ndarray

Parameters **i** – Index of column.

Returns Numpy array with data from column `i`.

get_row(*i: int*) → numpy.ndarray

Parameters **i** – Index of row.

Returns Numpy array with data from row `i`.

get_unit(*col: Union[int, str]*) → str
Get unit of column `col`.

Parameters **col** – Column index or name.

Returns Unit of column.

has_unit(*col: Union[int, str]*) → bool
Returns `True` column units have been set and `False` otherwise.

Parameters **col** – Column index or name.

Returns `True` if column_units have been set, `False` otherwise.

load(*filename: str*) → None
Loads data from csv or tab delimited tex file. The _header is searched for measurement parameters.

Parameters **filename** – Absolute or relative path of file to load.

property ncols
Number of columns of the ResultTable.

property nrows
Number of rows of the ResultTable.

plot(*x_clmn: int = 0*, *y_clmns: Optional[List[str]] = None*, *func: Callable = <function ResultTable.<lambda>>*, *live: bool = False*, ***kwargs*) → result_table.ResultTablePlot
Plots the data. This method should not be called from a thread. The column containing the x-axis data is specified (defaults to first column), all other data is plotted on the y-axis. This method requires Matplotlib to be installed and accepts, in addition to the arguments documented here, the same keyword arguments as `matplotlib.pyplot.plot()`.

Column titles are taken as legend labels. *plot()* tries to determine a common y-axis unit and name from all given labels.

Parameters

- **x_clmn** (*int or str*) – Integer or name of column containing the x-axis data.
- **y_clmns** – List of column numbers or column names for y-axis data. If not given, all columns will be plotted against the x-axis column.
- **func** – Function to apply to y-data before plotting.
- **live** – If `True`, update the plot when new data is added. Plotting will be carried out in the main (GUI) thread, therefore take care not to block the thread. This can be achieved for instance by adding data in a background thread which carries out the measurement, or by calling *matplotlib.pyplot.pause* after adding data to give the GUI time to update.

Returns *ResultTablePlot* instance with Matplotlib figure.

Raises **ImportError** – If import of matplotlib fails.

save(*filename: str*, *ext: str = '.txt'*) → None
Saves the result table to a text file. The file format is:

- The _header contains all measurement parameters as comments.
- Column titles contain column_names and column_units of measured quantity.
- Delimited columns contain the data.

Files are saved with the specified extension (default: '.txt'). The classes default delimiters are used to separate columns and rows.

Parameters

- **filename** – Path of file to save. Relative paths are interpreted with respect to the current working directory.
- **ext** – File extension. Defaults to '.txt'.

save_csv(*filename: str*) → None
Saves the result table to a csv file. The file format is:

- The _header contains all measurement parameters as comments.
- Column titles contain column_names and column_units of measured quantity.
- Comma delimited columns contain the data.

Files are saved with the extension '.csv' and other extensions are overwritten.

Parameters **filename** – Path of file to save. Relative paths are interpreted with respect to the current working directory.

set_unit(*col: Union[int, str], unit: str*) → None
Set unit of column `col`.

Parameters

- **col** – Column index or name.
- **unit** – Unit string.

property shape
A tuple representing the dimensionality of the ResultTable.

class result_table.**ResultTablePlot**(*result_table: result_table.ResultTable*, *x_clmn: int = 0*, *y_clmns: List[Union[str, int]] = None*, *func: Callable = <function ResultTablePlot.<lambda>>*, *live: bool = False*, ***kwargs*)
Bases: `object`

Plots the data from a given *ResultTable* instance. Axes labels are automatically generated from column titles and units. This class requires Matplotlib to be installed. In addition to the arguments documented here, class:*ResultTable* accepts the same keyword arguments as `matplotlib.pyplot.plot()`.

Parameters

- **result_table** (*ResultTable*) – *ResultTable* instance with data to plot.
- **x_clmn** (*int or str*) – Integer or name of column containing the x-axis data.
- **y_clmns** (*list(int or str)*) – List of column numbers or column names for y-axis data. If not given, all columns will be plotted against the x-axis column.
- **func** – Function to apply to y-data before plotting.
- **live** – If `True`, update the plot when new data is added (default: `False`). Plotting will be carried out in the main (GUI) thread, therefore take care not to block the thread. This can be achieved for instance by adding data in a background thread which carries out the measurement, or by calling *matplotlib.pyplot.pause* after adding data to give the GUI time to update.

show() → None
Shows the plot.

update() → None
Updates the plot with the data of the corresponding *ResultTable*. This will be called periodically when :param:`live` is `True`.

CHAPTER

THREE

CHANGELOG

3.1 v2.0.2

Plots of ResultTable data will now show a legend by default.

3.2 v2.0.1

Fixes an error where a voltage sweep would end with an AttributeError: cannot set property `busy`.

3.3 v2.0.0

This release completely overhauls how Keithley commands are generated. Instead of hard- coding available commands for a particular series or model of Keithley, all available commands are retrieved on demand from the Keithley itself. This is possible because the Keithley's TSP scripts use the Lua programming language which allows such introspection.

The main disadvantage of this approach is that most Keithley attributes will only be generated *after* connecting to an instrument. The main advantage is that all Keithley commands of all models which use TSP are supported and support for any future commands will be automatic. This removes the need to update the driver as the command set evolves, barring changes to the syntax, and enables automatic support for models with different command sets or a different number of SMUs. Furthermore, there have been issues in the past with missing commands and constants due to oversight. Those will no longer occur.

The second major change is a switch from camel-case to snake-case for the public API. For example, `Keithley2600.applyVoltage` has been renamed to `Keithley2600.apply_voltage`.

Other changes include:

- Type hints are used throughout.
- The Python syntax has been modernised for Python 3.6.
- The Keithley no longer beeps at the end of custom sweeps.
- Added API `Keithley2600.send_trigger` to send a trigger signal to the Keithley. This can be used to manually start a pre-programmed sweep.
- Added API `KeithleyClass.create_lua_attr` to create a new attribute in the Keithley namespace. Use `Keithley2600.create_lua_attr` to create global variables.
- Added API `KeithleyClass.delete_lua_attr` to delete an attribute from the Keithley namespace. Use `Keithley2600.delete_lua_attr` to delete global variables.

- `Keithley2600.connect()` now returns `True` if the connection was established and `False` otherwise.

3.4 v1.4.1

Changed:

- Replaced deprecated `visa` import with `pyvisa`.

3.5 v1.4.0

Added:

- Save time stamps with measurement data.

Changed:

- Renamed `ResultTablePlot.update_plot` to `ResultTablePlot.update`.
- Improved documentation of (live) plotting.
- Added `SENSE_LOCAL`, `SENSE_REMOTE` and `SENSE_CALA` to dictionary.

Fixed:

- Fixed explicitly defined methods such as `Keithley2600.applyVoltage` not appearing in dictionary.

3.6 v1.3.4

Fixed:

- Fixed a typo in the column labels of the dataset returned by `outputMeasurement`.

3.7 v1.3.3

Added:

- Added `__dir__` proprty to Keithley2600 and its classes to support autocompletion. The dictionary of commands is craeted from the Keithley reference manual.

Changed:

- Remember PyVisa connection settings which are passed as keyword arguments to `Keithley2600`. Previously, calling `Keithley2600.connect(...)` would revert to default settings.

Fixed:

- Explicitly set source mode in `Keithley2600.applyCurrent` and `Keithley2600.applyVoltage`.

3.8 v1.3.2

This release drops support for Python 2.7. Only Python 3.6 and higher are supported

Fixed:

- Fixed a bug in `rampToVoltage` where the target voltage would not be set correctly if it was smaller than the step size.

3.9 v1.3.1

Added:

- Optional argument `raise_keithley_errors`: If `True`, the Keithley's error queue will be checked after each command and any Keithley errors will be raised as Python errors. This causes significant communication overhead but facilitates the debugging of faulty scripts since an invalid command will raise a descriptive error instead of failing silently.

Fixed:

- Thread safety of communication with Keithley. `Keithley2600Base` now uses its own lock instead of relying on PyVisa's thread safety.

3.10 v1.3.0

This version includes some API changes and updates to the documentation and doc strings.

Added:

- Accept `range` (Python 2 and 3) and `xrange` (Python 2) as input for voltage sweep lists.

Changed:

- Methods `header` and `parse_header` of `ResultTable` are now private.
- Cleaned up and updated documentation.

Removed:

- Removed deprecated function `Keithley2600.clearBuffer()`. Use `buffer.clear()` and `buffer.clearcache()` instead where `buffer` is a Keithley buffer instance, such as `Keithley2600.smua.nvbuffer1`.

3.11 v1.2.2

Added:

- Added `shape` property to `ResultTable`.
- Added string representation of `ResultTable` which returns the first 7 rows as neatly formatted columns (similar to pandas dataframes).

3.12 v1.2.1

Fixed:

- Fixed a critical error when initializing and appending columns to an empty `ResultTable` instance.

3.13 v1.2.0

Added:

- New method `readErrorQueue` which returns a list of all errors in the Keithley's error queue.
- Support for Keithley TSP functions with multiple return values. Previously, only the first value would be returned.
- Added `ResultTablePlot` class to plot the data in a `ResultTable`.
- Added live plotting to `ResultTable` and its subclasses. Pass the keyword argument `live=True` to the `plot` method for the plot to update dynamically when new data is added.

Changed:

- Optimized I/O: Keithley function calls to only use a single `query` call instead of consecutive `query` and `read` calls.
- Emtpy strings returned by the Keithley will always be converted to `None`. This is necessary to enable the above change.
- Renamed `TransistorSweepData` to `FETResultTable`. Renamed `sweep_data` module to `result_table`.

Removed:

- Removed `IVSweepData`. The was no clear added value over using `ResultTable` directly.

3.14 v1.1.1

Fixed:

- Fixed a thread safety bug: Fixed a bug that could cause the wrong result to be returned by a query when using `Keithley2600` from multiple threads.

3.15 v1.1.0

Added:

- Sphinx documentation.

3.16 v1.0.0

Added:

- Added the base class `ResultTable` to store, save and load tables of measurement data together with column titles, units, and measurement parameters. The data is stored internally as a 2D numpy array and can be accessed in a dictionary-type fashion with column names as keys. Additionally, `ResultTable` provides a basic plotting method using matplotlib.

Changed:

- **`TrasistorSweepData` and `IVSweepData` now inherit from `ResultTable` and have been** significantly simplified. Formats for saving and loading the data to files have slightly changed:
  - The line with column headers is now marked as a comment and starts with '#'.
  - All given measurement parameters are saved in the file's _header. Specifically, `TrasistorSweepData.load()` expects the parameter `sweep_type` to be present in the _header and have one of the values: 'transfer' or 'output'.
  - Options to read and write in CSV format instead of tab-delimited columns are given.

  As a result, data files created by versions < 1.0.0 need to be modified as follows to be recognized:
  - Prepend '#' to the line with column titles.
  - Add the line '# sweep_type: type' to the _header where type can be 'transfer', 'output', or 'iv'.

Removed:

- `clearBuffers` method from `Keithley2600` has been deprecated. Clear the buffers directly with `buffer.clear()` instead, where `buffer` is a keithley buffer instance such as `k.smua.nvbuffer1`.

3.17 v0.3.0

Added:

- `Keithley2600` methods now accept `Keithley2600` objects as arguments, for instance, one can now write

```
# assume we have a Keithley2600 instance 'k'
k.smua.measureiv(k.smua.nvbuffer1, k.smua.nvbuffer2)
```

  instead of needing to use their string representation:

```
k.smua.measureiv('smua.nvbuffer1', 'smua.nvbuffer2')
```

- Keyword arguments can now be given to `Keithley2600()` and will be passed on to the visa resource (e.g., `baud_rate=9600`).

Changed:

- Code simplifications resulting from the above.
- `k.readBuffer(buffer)` no longer clears the given buffer.
- When attempting to create a new instance of `Keithley2600` with the name VISA address as an existing instance, the existing instance is returned instead.

Removed:

- `k.clearBuffers(...)` now logs a deprecation warning and will be removed in v1.0. Clear the buffers directly with `buffer.clear()` instead.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

r

INDEX

T

U

V