# Contents

1 keithley2600
   1.1 About ................................................................. 3
   1.2 Installation .......................................................... 3
   1.3 Usage ................................................................. 4
   1.4 Backend selection .................................................. 5
   1.5 System requirements ............................................... 5
   1.6 Documentation ...................................................... 6

2 Modules ................................................................. 7
   2.1 Keithley driver ....................................................... 7
   2.2 Classes to hold sweep results .................................... 12

3 Changelog ............................................................... 19
   3.1 v1.4.0 ................................................................. 19
   3.2 v1.3.4 ................................................................. 19
   3.3 v1.3.3 ................................................................. 19
   3.4 v1.3.2 ................................................................. 20
   3.5 v1.3.1 ................................................................. 20
   3.6 v1.3.0 ................................................................. 20
   3.7 v1.2.2 ................................................................. 20
   3.8 v1.2.1 ................................................................. 21
   3.9 v1.2.0 ................................................................. 21
   3.10 v1.1.1 ................................................................. 21
   3.11 v1.1.0 ................................................................. 21
   3.12 v1.0.0 ................................................................. 22
   3.13 v0.3.0 ................................................................. 22

4 Indices and tables ..................................................... 25
   Python Module Index .................................................. 27
   Index ........................................................................... 29
This documentation provides both a quick introduction to keithley2600 and a detailed reference of the API:
A full Python driver for the Keithley 2600 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

keithley2600 provides access to base functions 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 functions, which have a syntax similar to Python.

**Warning:** There are currently only heuristic checks for allowed commands and arguments by the driver itself. See the Keithley 2600 reference manual for all available commands. To enable command checking, set the keyword argument `raise_keithley_errors = True` in the constructor. When `raise_keithley_errors` is True, all invalid commands will be raised as Python errors. This is done by reading the Keithley’s error queue after every command and will therefore result in significant communication overhead.

Almost all Keithley TSP commands can be used with this driver. Not supported are:

- All Keithley IV sweep commands. We implement our own in the Keithley2600 class.
- Keithley TSP functions that have the same name as a Keithley TSP attribute (and vice versa). The driver cannot decide whether to handle them as a function call or as an attribute access. Currently, there is only one such case:
  - `io.output()` has been dropped because it conflicts with `smuX.source.output`, which is more commonly used.
- Keithley TSP commands that have the same name as built-in attributes of the driver. Currently, this is only:
  - `lan.trigger[N].connected: conflicts with the attribute Keithley2600.connected`.

1.2 Installation

Install the stable version from PyPi by running
1.3 Usage

Connect to the Keithley 2600 and perform some base commands:

```python
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:

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

k.setIntegrationTime(k.smua, 0.001)  # sets integration time in sec
k.applyVoltage(k.smua, 10)  # turns on and applies 10V to SMUA
k.applyCurrent(k.smub, 0.1)  # sources 0.1A from SMUB
k.rampToVoltage(k.smua, 10, delay=0.1, stepSize=1)  # ramps SMUA to 10V in steps of 1V

# sweep commands
k.voltageSweepSingleSMU(k.smua, range(0, 61), t_int=0.1,
delay=-1, pulsed=False)
k.voltageSweepDualSMU(sm1=k.smua, sm2=k.smub, sm1_sweeplist=range(0, 61),
sm2_sweeplist=range(0, 61), t_int=0.1, delay=-1, pulsed=False)
k.transferMeasurement( ... )
k.outputMeasurement( ... )
```

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:

```python
>>> 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 `voltageSweepSingleSMU` and `voltageSweepDualSMU` return lists with the measured voltages and currents. The higher level commands `transferMeasurement` and `outputMeasurement` 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:

```python
import time
from keithley2600 import Keithley2600, ResultTable

k = Keithley2600('TCP/IP0::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.applyVoltage(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 proprietary 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:

```python
from keithley2600 import Keithley2600

k = Keithley2600('TCP/IP0::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 and 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 and higher
1.6 Documentation

- API documentation of keithley2600: https://keithley2600.readthedocs.io/en/latest/
This section gives an overview of keithley2600’s modules:

## 2.1 Keithley driver

Core driver with the low level functions.

```python
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, visa_library='@py', raise_keithley_errors=False, **kwargs)
    Bases: keithley_driver.MagicClass
    Keithley2600 driver

    Keithley driver for base functionality. It replicates the functionality and syntax from the Keithley TSP commands, which have a syntax similar to python. Attributes are created on-access if they correspond to Keithley TSP type commands.

    Parameters
    
    * `visa_address (str)` – Visa address of the instrument.
    * `visa_library (str)` – 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** *(bool)* – 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 baud-rate or timeout. If not given, reasonable defaults will be used.

**Variables**

• **connection** – Attribute holding a reference to the actual connection.

• **connected**(bool) – True if connected to an instrument, False otherwise.

• **busy**(bool) – True if a measurement is running, False otherwise.

• **TO_TSP_LIST**(list) – List of python types which will be converted to Keithley TSP lists by this driver and can be used as inputs. Currently, those are list, numpy.ndarray, tuple, set, and range (xrange in Python 2).

• **CHUNK_SIZE**(int) – 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. Almost all Keithley TSP commands can be used with this driver. Not supported are:

• `lan.trigger[N].connected`: conflicts with the connected attribute

• `io.output()`: conflicts with `smuX.source.output`

• All Keithley IV sweep commands. We implement our own in *Keithley2600*.

**Examples**

```python
>>> 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**)**

Connects to Keithley.

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

**disconnect()**

Disconnects from Keithley.

**class keithley_driver.Keithley2600(visa_address, visa_library='@py', raise_keithley_errors=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**(str) – Visa address of the instrument.

• **visa_library**(str) – 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** *(bool)* – 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 baud-rate or timeout. If not given, reasonable defaults will be used.

Variables

• **connection** – Attribute holding a reference to the actual connection.

• **connected** *(bool)* – True if connected to an instrument, False otherwise.

• **busy** *(bool)* – True if a measurement is running, False otherwise.

**Examples**  
**Base commands from Keithley TSP:**

```python
>>> k = Keithley2600('TCPIP0::192.168.2.121::INSTR')
>>> volts = k.smua.measure.v()  # measures and returns the smuA voltage
>>> k.smua.source.levelv = -40  # sets source level of smuA
>>> k.smua.nvbuffer1.clear()  # clears nvbuffer1 of smuA
```

**New mid-level commands:**

```python
>>> data = k.readBuffer(k.smua.nvbuffer1)
>>> errs = k.readErrorQueue()
>>> k.setIntegrationTime(k.smua, 0.001)  # in sec

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

**# voltage sweeps, single and dual SMU**

```python
>>> k.voltageSweepSingleSMU(smu=k.smua, smu_sweeplist=range(0, 61),
  t_int=0.1, delay=-1, pulsed=False)
>>> k.voltageSweepDualSMU(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:**

```python
>>> data1 = k.outputMeasurement(...)  # records output curve
>>> data2 = k.transferMeasurement(...)  # records transfer curve
```

**readErrorQueue()**

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.

**Return type** list

**static readBuffer**(buffer)

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.
Return type  list

setIntegrationTime (smu, t_int)
Sets the integration time of SMU for measurements in sec.

Parameters

• smu – A keithley smu instance.
• t_int (float) – 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.

applyVoltage (smu, voltage)
Turns on the specified SMU and applies a voltage.

Parameters

• smu – A keithley smu instance.
• voltage (float) – Voltage to apply in Volts.

applyCurrent (smu, curr)
Turns on the specified SMU and sources a current.

Parameters

• smu – A keithley smu instance.
• curr (float) – Current to apply in Ampere.

measureVoltage (smu)
Measures a voltage at the specified SMU.

Parameters smu – A keithley smu instance.

Returns Measured voltage in Volts.

Return type  float

measureCurrent (smu)
Measures a current at the specified SMU.

Parameters smu – A keithley smu instance.

Returns Measured current in Ampere.

Return type  float

rampToVoltage (smu, target_volt, delay=0.1, step_size=1)
Ramps up the voltage of the specified SMU. Beeps when done.

Parameters

• smu – A keithley smu instance.
• target_volt (float) – Target voltage in Volts.
• step_size (float) – Size of the voltage steps in Volts.
• delay (float) – Delay between steps in sec.

voltageSweepSingleSMU (smu, smu_sweeplist, t_int, delay, pulsed)
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 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** Lists of voltages and currents measured during the sweep (in Volt and Ampere, respectively): \((v\_smu, i\_smu)\).

**Return type** \((list, list)\)

---

```python
voltageSweepDualSMU(sm1, smu2, smu1_sweeplist, smu2_sweeplist, t_int, delay, pulsed)
```

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 range / xrange).

• **smu2_sweeplist** – Voltages to sweep at smu2 (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** Lists of voltages and currents measured during the sweep (in Volt and Ampere, respectively): \((v\_smu1, i\_smu1, v\_smu2, i\_smu2)\).

**Return type** \((list, list, list, list)\)

---

```python
transferMeasurement(sm_u_gate, sm_u_drain, vg_start, vg_stop, vg_step, vd_list, t_int, delay, pulsed)
```

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

**Parameters**

• **sm_u_gate** – Keithley smu attached to gate electrode.

• **sm_u_drain** – Keithley smu attached to drain electrode.

• **vg_start (float)** – Start voltage of transfer sweep in Volt.

• **vg_stop (float)** – End voltage of transfer sweep in Volt.

• **vg_step (float)** – 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.
• **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 Transfer curve data.

Return type `sweep_data.TransistorSweepData`

`outputMeasurement(smGate, smDrain, vStart, vStop, vStep, vgList, t_int, delay, pulsed)`

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.

• **v_start** (*float*) – Start voltage of output sweep in Volt.

• **v_stop** (*float*) – End voltage of output sweep in Volt.

• **v_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.

Return type `sweep_data.TransistorSweepData`

`playChord(notes=\['C6', 'E6', 'G6'\], durations=0.3)`

Plays a chord on the Keithley.

Parameters

• **notes** (*list*) – 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** (*float or list*) – 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.

2.2 Classes to hold sweep results

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

```python
class result_table.ColumnTitle(name, unit=None, unit_fmt='{}')
    Bases: object
    Class to hold a column title.
```

Chapter 2. Modules
Parameters

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

```python
class result_table.ResultTable (column_titles=None, units=None, data=None, params=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.ndarray or NoneType)* – 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** *(dict)* – Dictionary of measurement parameters.

Examples

Create a `ResultTable` to hold current-vs-time data:

```python
>>> 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'], pars)

# create a live plot of the data
>>> fig = rt.plot(live=True)

Create a Keithley2600 instance and record some data:

```python
>>> 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.applyVoltage(k.smua, 10)
...     i = k.smua.measure.i()
...     rt.append_row([v, i])
...     time.sleep(1)

Print a preview of data to the console:

```python
>>> print(rt)
Voltage [V]  Current [A]
0.00000e+00  1.0232e-04
1.00000e+00  2.2147e-04
2.00000e+00  3.6077e-04
3.00000e+00  5.2074e-04
4.00000e+00  6.9927e-04
```
Save the recorded data to a tab-delimited text file:

```python
>>> rt.save '~/Desktop/stress_test.txt'
```

**nrows**
Number of rows of the ResultTable.

**ncols**
Number of columns of the ResultTable.

**shape**
A tuple representing the dimensionality of the ResultTable.

**column_names**
List of strings with column names.

**column_units**
List of strings with column units.

**has_unit** *(col)*
Returns True column units have been set and False otherwise.

```
Parameters col (int or str) -- Column index or name.
Returns True if column_units have been set, False otherwise.
Return type bool
```

**get_unit** *(col)*
Get unit of column col.

```
Parameters col (int or str) -- Column index or name.
Returns Unit string.
Return type str
```

**set_unit** *(col, unit)*
Set unit of column col.

```
Parameters col (int or str) -- Column index or name.
unit (str) -- Unit string.
```

**clear_data** ()
Clears all data.

**append_row** *(data)*
Appends a single row to the data array.

```
Parameters data -- Iterable with the same number of elements as columns in the data array.
```

**append_rows** *(data)*
Appends multiple rows to the data array.

```
Parameters data -- List of lists or numpy array with dimensions matching the data array.
```

**append_column** *(data, name, unit=None)*
Appends a single column to the data array.

```
Parameters data -- Iterable with the same number of elements as rows in the data array.
name (str) -- Name of new column.
```
• `unit (str)` – Unit of values in new column.

`append_columns (data, column_titles, units=None)`

Appends multiple columns to data array.

Parameters

• `data (list)` – List of columns to append.
• `column_titles (list)` – List of column titles (strings).
• `units (list)` – List of units for new columns (strings).

`get_row (i)`

Returns Numpy array with data from row i.

Return type `numpy.ndarray`

`get_column (i)`

Returns Numpy array with data from column i.

Return type `numpy.ndarray`

`save (filename, ext='.txt')`

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 (str)` – Path of file to save. Relative paths are interpreted with respect to the current working directory.
• `ext (str)` – File extension. Defaults to `.txt`.

`save_csv (filename)`

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 (str)` – Path of file to save. Relative paths are interpreted with respect to the current working directory.

`load (filename)`

Loads data from csv or tab delimited text file. The _header is searched for measurement parameters.

Parameters `filename (str)` – Absolute or relative path of file to load.

`plot (x_clmn=0, y_clmns=None, func=<function ResultTable.<lambda>>, live=False, **kwargs)`

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_cimns (list)` – 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)` – Function to apply to y-data before plotting.
- `live (bool)` – 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.

**Returns** `ResultTablePlot` instance with Matplotlib figure.

**Return type** `ResultTablePlot`

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

```python
class result_table.FETResultTable(column_titles=None, units=None, data=None, params=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)

   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.

   Return type `ResultTablePlot`

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

```python
class result_table.ResultTablePlot(result_table, x_clmn=0, y_clmns=None, func=<function ResultTablePlot.<lambda>>, live=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)` – Function to apply to y-data before plotting.
- `live (bool)` – 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()
        Shows the plot.
update()
        Updates the plot with the data of the corresponding `ResultTable`. This will be called periodically when :param:`live` is `True`.  
```
3.1 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.

3.2 v1.3.4

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

3.3 v1.3.3

Added:
- Added `__dir__` property to `Keithley2600` and its classes to support autocompletion. The dictionary of commands is created 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.

3.4 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.

3.5 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.

3.6 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.

3.7 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).

### 3.8 v1.2.1

**Fixed:**

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

### 3.9 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.
• Empty 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`. There was no clear added value over using `ResultTable` directly.

### 3.10 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.

### 3.11 v1.1.0

**Added:**

• Sphinx documentation.
3.12 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`.

3.13 v0.3.0

Added:

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

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

  instead of needing to use their string representation:

  ```python
  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 4

Indices and tables

- genindex
- modindex
- search
Python Module Index

k
keithley_driver, 7

r
result_table, 12
<table>
<thead>
<tr>
<th>A</th>
<th>append_column() (result_table.ResultTable method), 14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>append_columns() (result_table.ResultTable method), 15</td>
</tr>
<tr>
<td></td>
<td>append_row() (result_table.ResultTable method), 14</td>
</tr>
<tr>
<td></td>
<td>append_rows() (result_table.ResultTable method), 14</td>
</tr>
<tr>
<td></td>
<td>applyCurrent() (keithley_driver.Keithley2600 method), 10</td>
</tr>
<tr>
<td></td>
<td>applyVoltage() (keithley_driver.Keithley2600 method), 10</td>
</tr>
<tr>
<td>C</td>
<td>clear_data() (result_table.ResultTable method), 14</td>
</tr>
<tr>
<td></td>
<td>column_names (result_table.ResultTable attribute), 14</td>
</tr>
<tr>
<td></td>
<td>column_units (result_table.ResultTable attribute), 14</td>
</tr>
<tr>
<td></td>
<td>ColumnTitle (class in result_table), 12</td>
</tr>
<tr>
<td></td>
<td>connect() (keithley_driver.Keithley2600Base method), 8</td>
</tr>
<tr>
<td>D</td>
<td>disconnect() (keithley_driver.Keithley2600Base method), 8</td>
</tr>
<tr>
<td>F</td>
<td>FETResultTable (class in result_table), 16</td>
</tr>
<tr>
<td>G</td>
<td>get_column() (result_table.ResultTable method), 15</td>
</tr>
<tr>
<td></td>
<td>get_row() (result_table.ResultTable method), 15</td>
</tr>
<tr>
<td></td>
<td>get_unit() (result_table.ResultTable method), 14</td>
</tr>
<tr>
<td>H</td>
<td>has_unit() (result_table.ResultTable method), 14</td>
</tr>
<tr>
<td>K</td>
<td>Keithley2600 (class in keithley_driver), 8</td>
</tr>
<tr>
<td></td>
<td>Keithley2600Base (class in keithley_driver), 7</td>
</tr>
<tr>
<td></td>
<td>keithley_driver (module), 7</td>
</tr>
<tr>
<td>L</td>
<td>KeithleyError, 7</td>
</tr>
<tr>
<td></td>
<td>KeithleyIOError, 7</td>
</tr>
<tr>
<td></td>
<td>load() (result_table.ResultTable method), 15</td>
</tr>
<tr>
<td>M</td>
<td>measureCurrent() (keithley_driver.Keithley2600 method), 10</td>
</tr>
<tr>
<td></td>
<td>measureVoltage() (keithley_driver.Keithley2600 method), 10</td>
</tr>
<tr>
<td>N</td>
<td>(ncols (result_table.ResultTable attribute), 14</td>
</tr>
<tr>
<td></td>
<td>nrows (result_table.ResultTable attribute), 14</td>
</tr>
<tr>
<td>O</td>
<td>outputMeasurement() (keithley_driver.Keithley2600 method), 12</td>
</tr>
<tr>
<td>P</td>
<td>playChord() (keithley_driver.Keithley2600 method), 12</td>
</tr>
<tr>
<td></td>
<td>plot() (result_table.FETResultTable method), 16</td>
</tr>
<tr>
<td></td>
<td>plot() (result_table.ResultTable method), 15</td>
</tr>
<tr>
<td>R</td>
<td>rampToVoltage() (keithley_driver.Keithley2600 method), 10</td>
</tr>
<tr>
<td></td>
<td>readBuffer() (keithley_driver.Keithley2600 static method), 9</td>
</tr>
<tr>
<td></td>
<td>readErrorQueue() (keithley_driver.Keithley2600 method), 9</td>
</tr>
<tr>
<td></td>
<td>result_table (module), 12</td>
</tr>
<tr>
<td></td>
<td>ResultTable (class in result_table), 13</td>
</tr>
<tr>
<td></td>
<td>ResultTablePlot (class in result_table), 16</td>
</tr>
<tr>
<td>S</td>
<td>save() (result_table.ResultTable method), 15</td>
</tr>
</tbody>
</table>
save_csv() (result_table.ResultTable method), 15
set_unit() (result_table.ResultTable method), 14
setIntegrationTime() (keithley_driver.Keithley2600 method), 10
shape (result_table.ResultTable attribute), 14
show() (result_table.ResultTablePlot method), 17

T
transferMeasurement() (keithley_driver.Keithley2600 method), 11

U
update() (result_table.ResultTablePlot method), 17

V
voltageSweepDualSMU() (keithley_driver.Keithley2600 method), 11
voltageSweepSingleSMU() (keithley_driver.Keithley2600 method), 10