

---

# **Skippylab Documentation**

*Release 0.7.0*

**Achim Stoessl**

**May 09, 2018**



---

## Contents

---

<b>1 Skippylab documentation contents</b>	<b>1</b>
1.1 skippylab package . . . . .	1
<b>2 Indices and tables</b>	<b>15</b>
<b>Python Module Index</b>	<b>17</b>



---

## SkippyLab documentation contents

---

### 1.1 skippyLab package

#### 1.1.1 Subpackages

##### skippyLab.instruments package

###### Submodules

##### skippyLab.instruments.function\_generators module

Function generators

```
class skippyLab.instruments.function_generators.Agilent33220AFunctionGenerator (ip='10.25.123.11',  
                                                                           gpiB_address=15,  
                                                                           loglevel=20)
```

Bases: `object`

##### skippyLab.instruments.gpiBcontrollers module

Wrapper around the PrologixGPIBEthernet adapter, to transparently manage instruments which are connected via GPIB and the respective controller

```
skippyLab.instruments.gpiBcontrollers.prologix_gpiB_ethernet_provider (ip,  
                                                                           gpiB_adddres)
```

Provide a vx11 compatible instrument which is accesible transparently through its ip.

###### Parameters

- **ip** (*str*) – ip address of the controller
- **gpiB\_address** (*str*) – gpiB adress of the instrument

**Returns** vx11.instrument

## skippylab.instruments.oscilloscopes module

Communicate with oscilloscope via vx11 protocol over LAN network

```
class skippylab.instruments.oscilloscopes.AbstractBaseOscilloscope (ip='169.254.68.19',  
loglevel=20)
```

Bases: `object`

A oscilloscope with a high sampling rate in the order of several gigasamples. Defines the scope API the DAQ relies on

**ACQUIRE\_ONE** = 'SEQ'

**CONTINUOUS\_RUN** = 'RUNSTop'

**MAXTRIALS** = 5

**acquire\_waveform**()

**ping**()

Check if oscilloscope is connected

**reopen\_socket**()

Close and reopen the socket after a timeout

**Returns** None

**samplingrate**

Get the current sampling rate

**Returns** float (GSamples/sec)

**select\_channel** (*channel*)

Select a channel for the data acquisition

**Parameters** **channel** (*int*) – Channel number

**Returns** None

**string\_encoding** = 'iso-8859-1'

```
class skippylab.instruments.oscilloscopes.RhodeSchwarzRTO1044 (ip)
```

Bases: `skippylab.instruments.oscilloscopes.AbstractBaseOscilloscope`

Made by Rhode&Schwarz, scope with sampling rate up to 20GSamples/s

**acquire\_waveform**()

Get the voltage values for a single waveform

**Returns** np.ndarray

**do\_single\_acquisition**()

**run**()

Start continuous acquisitions

Returns:

**samplingrate**

Get the current sampling rate

**Returns** float (GSamples/sec)

**select\_channel** (*channel*)

Select the channel for the readout.

**Parameters** **channel** (*int*) – Channel number (1-4)

**Returns** None

**stop()**

**triggerrate**

Get the triggerrate of the scope

**Parameters** **interval** (*float*) – measurement time in seconds to

**Returns** float

**class** `skippylab.instruments.oscilloscopes.TektronixDPO4104B(ip, loglevel=20)`

Bases: `skippylab.instruments.oscilloscopes.AbstractBaseOscilloscope`

Oscilloscope of type DPO4104B manufactured by Tektronix

**acquire**

**acquire\_mode**

**acquire\_waveform** (*header=None, return\_digitizer\_levels=False*)

Get the waveform data

**Keyword Arguments**

- **header** (*dict*) – if header is None, a new header will be acquired
- **return\_digitizer\_levels** (*bool*) – return the waveform data in digitizer levels, not volts. Saves space for storage, since int8 can be used for 1-bit representation. ..and there is no float8 for obvious reasons.

**Returns** np.ndarray

**average\_waveform** (*n=10*)

Acquire some waveforms and take the average

**Keyword Args.** **n** (int): number of waveforms to average over

**Returns** tuple(np.array). xs, ys

**binary\_formats** = {'RI': '!b'}

**binary\_header\_pattern** = `re.compile('#(?:P<bin_head>[0-9]*)')`

**static** **convert\_waveform** (*header, waveform*)

**data**

**data\_start**

**data\_stop**

**data\_width**

**static** **decode\_ascii\_waveform** (*response*)

Search the response for waveform data when in ascii format

**Parameters** **response** (*str*) – Hopefully the result of a CURVE command or similar

**Returns** np.ndarray

**decode\_binary\_waveform** (*response, header*)

Decaode a waveform in binary format. To do so, the header is required to know about the exact format.

**Parameters**

- **response** (*str*) – Hopefully the response to some CURVE command or similar

- **header** (*dict*) – A parsed waveform header

**Returns** np.ndarray

**static decode\_header** (*response*, *return\_last\_index=False*, *absolute\_timing=False*)

Parse a response searching for waveform header data

**Parameters** **head** (*str*) – hopefully the result of some WAVFrm or similar command

**Keyword Arguments**

- **return\_last\_index** (*bool*) – if True, also the last index of the header in the string will be returned
- **absolute\_timing** (*bool*) – try to infer the absolute timing (whatever that means) # FIXME!

**Returns** dict/tuple

**fill\_buffer** ()

Returns:

**histbox**

**histend**

**histogram**

Return a histogram which might be recorded by the scope

**histstart**

**make\_n\_acquisitions** (*n*, *trials=20*, *return\_only\_charge=False*, *single\_acquisition=True*, *return\_digitizer\_levels=False*)

Acquire n waveforms

**Parameters** **n** (*int*) – Number of waveforms to acquire

**Keyword Arguments**

- **trials** (*int*) – Set breaking condition when to abort acquisition
- **return\_only\_charge** (*bool*) – don't get the wf, but only integrated charge instead
- **single\_acquisition** (*bool*) – use the scopes single acquisition mode
- **return\_digitizer\_levels** (*bool*) – return the waveform data in digitizer levels, not volts. Saves space for storage, since int8 can be used for 1-bit representation. ..and there is no float8 for obvious reasons.

**Returns** [wf\_1,wf\_2,...]

**Return type** list

**pull** (*buff\_header=True*, *use\_buffered\_acq\_window=True*, *use\_channel\_info=True*)

Fit in the API for the DAQ. Returns waveform data

**Keyword Arguments**

- **buff\_header** (*bool*) – buffer the header for subsequent acquisition without changing the parameters of the acquisition (much faster)
- **Default value of this should be False, however requires DAQ API change (FIXME!)** –
- **use\_buffered\_acq\_window** (*bool*) – set this flag to cache the length of the acquisition window internally so that it does not get resetted when switching channels
- **use\_channel\_info** (*bool*) – select the channel on each submit



**Returns** dict

**reset\_acquisition\_window** ()

Reset the acquisition window to the maximum possible acquisition window Returns:In

None

**samplingrate**

The samplingrate in GSamples/S

**Returns** float

**select\_channel** (*channel*)

Select the channel for the readout

**Parameters** **channel** (*int*) – Channel number (1-4)

**Returns** None

**set\_acquisition\_window** (*start, stop*)

Set the acquisition window in bin number

**Parameters**

- **start** (*int*) – start bin
- **stop** (*int*) – stop bin

**Returns** None

**set\_acquisition\_window\_from\_internal\_buffer** ()

Use the internal buffer to set the data acquisition window. Might be necessary if the channel was switched in the meantime

**Returns** None

**set\_feature\_acquisition\_window** (*leading, trailing, n\_waveforms=20*)

Set the acquisition window around the most prominent feature in the waveform

**Parameters**

- **leading** (*float*) – leading ns before the most prominent feature
- **trailing** (*float*) – trailing ns after the most prominent feature

**Keyword Args** **n\_waveforms** (*int*): average over *n\_waveforms* to identify the most prominent feature

**Returns** None

**set\_waveform\_encoding** (*enc*)

Define the waveform encoding

**Parameters** **enc** –

Returns:

**show\_waveforms** (*n=5*)

Demonstration function: Will use pylab show to plot some acquired waveforms

**Keyword Arguments** **n** (*int*) – number of waveforms to show

**Returns** None

**source**

**time\_binwidth**

Get the binwidth of the time - that is sampling rate

**Returns** float

**trigger\_continuous** ()

**trigger\_frequency\_enabled**

**trigger\_single** ()

**triggerrate**

The rate the scope is triggering. The scope in principle provides this number, however we have to work around it as it does not work reliably

**Keyword Arguments** **interval** (*int*) – time interval to integrate measurement over in seconds

**Returns** float

**waveform\_bins**

Get the time bin numbers for the waveform voltage data

**Returns** np.ndarray

**waveform\_enc**

**waveform\_times**

Get the time for the waveform bins

**Returns** np.ndarray

**class** `skippyLab.instruments.oscilloscopes.UnknownOscilloscope` (*ip='169.254.68.19', loglevel=20*)

Bases: `skippyLab.instruments.oscilloscopes.AbstractBaseOscilloscope`

Use for testing and debugging

**acquire\_waveform** ()

**samplingrate** ()

Get the current sampling rate

**Returns** float (GSamples/sec)

**select\_channel** (*channel*)

Select a channel for the data acquisition

**Parameters** **channel** (*int*) – Channel number

**Returns** None

**class** `skippyLab.instruments.oscilloscopes.Waveform` (*header, raw\_waveform*)

Bases: `object`

A non-oscilloscope dependent representation of a measured waveform

**load** ()

**save** ()

**time\_bins** ()

**volts** ()

`skippyLab.instruments.oscilloscopes.get_header` (*self*)

`skippyLab.instruments.oscilloscopes.set_header` (*self, header*)

`skippylab.instruments.oscilloscopes.setget (command)`

Shortcut to construct property object to wrap getters and setters for a number of settings

**Parameters**

- **command** (*str*) – The command being used to get/set. Get will be a query
- **value** (*str*) – The value to set

**Returns** property object

## skippylab.instruments.powersupplies module

Connection to power supply unit

```
class skippylab.instruments.powersupplies.KeysightE3631APowerSupply (ip='10.25.124.252',
                                                                    gpib_address=5,
                                                                    loglevel=20)
```

Bases: `object`

A low voltage power supply with two channels, +6V and +- 25V manufactured by Keysight. The power supply does not have an ethernet port, so the connection is done via GPIB and a prologix GPIB Ethernet connector

**error\_state**

Read out the error register of the power supply

**Returns** str

**measure\_current (channel)**

Measure current on given channel

**Parameters** **channel** (*str*) –

**Returns** float

**off ()**

Cut the power on all channels

**Returns** None

**on ()**

Enable power on all channels

**Returns** None

**output**

**ping ()**

Check the connection

**Returns** str

**select\_channel (channel)**

Select either the +6, +25 or -25V channel

**Parameters** **channel** (*str or int*) –

**Returns** None

**set\_voltage (channel, voltage)**

Set the supplied voltage of a channel to the desired value

**Parameters**

- **channel** (*str or int*) –

- `voltage` (*float*) –  
Returns None

## Module contents

### skippylab.scpi package

#### Submodules

#### skippylab.scpi.commands module

A namespace for oscilloscope string commands. The commands are send as ASCII to the scope using a socket connection

```
class skippylab.scpi.commands.KeysightE3631APowerSupplyCommands
```

```
    Bases: object
```

```
    Namespace for the commands of the KeysightE3631APowerSupply
```

```
    APPLY = 'APPLY'
```

```
    CHANNEL = 'INST'
```

```
    CURRENT = 'CURRENT'
```

```
    DC = 'DC'
```

```
    ERROR_STATEQ = 'SYST:ERR?'
```

```
    MEASURE = 'MEASURE'
```

```
    N25 = 'N25V'
```

```
    OFF = 'OFF'
```

```
    ON = 'ON'
```

```
    OUTPUT = 'OUTPUT:STATE'
```

```
    P25 = 'P25V'
```

```
    P6 = 'P6V'
```

```
    VOLT = 'VOLT'
```

```
class skippylab.scpi.commands.RhodeSchwarzRTO1044Commands
```

```
    Bases: object
```

```
    Namespace for the commands for the RhodeSchwarz oscilloscope
```

```
    CH1 = 'CHAN1'
```

```
    CH2 = 'CHAN2'
```

```
    CH3 = 'CHAN3'
```

```
    CH4 = 'CHAN4'
```

```
    CURVE = 'DATA:VALues?'
```

```
    N_ACQUISITONS = 'ACquire:CURRent?'
```

```
    RUN = 'RUN'
```

```

SINGLE = 'SINGLE'
STOP = 'STOP'
WAVEFORM = 'DATA?'
WF_HEADER = 'DATA:HEADer?'

```

```

class skippylab.scpi.commands.TektronixDPO4104BCommands
    Bases: object

```

Namespace for the commands for the TektronixDP04104B

```

ACQUISITION_MODE = 'ACquire:STOPAfter'
CH1 = 'CH1'
CH2 = 'CH2'
CH3 = 'CH3'
CH4 = 'CH4'
N_ACQUISITIONS = 'ACquire:NUMACq?'
OFF = 'OFF'
ON = 'ON'
ONE = '1'
TRIGGER_FREQUENCYQ = 'TRIGger:FREQuency?'
TRIGGER_FREQUENCY_ENABLED = 'DISplay:TRIGFrequency'
WF = 'WAVFrm?'
WF_BYTEWIDTH = 'DATA:WIDTH'
WF_HEADER = 'WFMOutpre:BYT_Nr?; :WFMOutpre:ENCdg?; :WFMOutpre:NR_Pt?; :WFMOutpre:XZEr?'
WF_NOHEAD = 'CURVE?'
ZERO = '0'

```

```

skippylab.scpi.commands.add_arg(cmd, arg)
    Add an argument to a command string. Concat the two.

```

#### Parameters

- **cmd** (*str*) – The base command
- **arg** (*str*) – An argument

**Returns** *str*

```

skippylab.scpi.commands.clean_response(response)
    Remove some EOL remainders from the resulting scope response

```

**Parameters** **response** (*str*) – response from the scope

**Returns** *str*

```

skippylab.scpi.commands.concat(*cmd)
    Combine several commands

```

**Parameters** **cmd** – list of commands

**Returns** *str*

`skippylab.sspi.commands.histbox_coordinates` (*left, top, right, bottom*)

Create a string for the box coordinates for the histogram set up by the scope itself. The result can be send to the scope to set the box.

**Parameters**

- **left** (*int*) –
- **top** (*int*) –
- **right** (*int*) –
- **bottom** (*int*) –

**Returns** *str*

`skippylab.sspi.commands.parse_curve_data` (*header, curve*)

Make sense out of that what is returned by CURVE. This works only if the scope is set to return the data in ASCII, not binary.

**Parameters**

- **header** (*dict*) – a parsed header
- **curv** (*str*) – returned by CURVE?

**Returns** `np.ndarray` *xs*, `np.ndarray` *values*

**Return type** *tuple*

`skippylab.sspi.commands.query` (*cmd*)

## Module contents

### 1.1.2 Submodules

### 1.1.3 skippylab.daq module

Use the scope as a DAQ

**class** `skippylab.daq.DAQ`

Bases: `object`

A virtual DAQ using an oscilloscope

**acquire** (*\*pullargs, \*\*pullkwargs*)

Go through the instrument list and trigger their pull methods to build an event

**Keyword Arguments** **\*\*pullkwargs** (*dict*) – will be passed on the individual pull methods

**Returns** `pyosci.Event`

**acquire\_n\_events** (*n\_events, trigger\_hook=<function DAQ.<lambda>>, trigger\_hook\_args=(None, ), pull\_args=(), pull\_kwargs={}*)

Continuous acquisition. Acquires *n* events. Yields events. Use trigger hook to define a function to decide when data is returned.

**Parameters**

- **n\_events** (*int*) – Number of events to acquire
- **trigger\_hook** (*callable*) – Trigger condition
- **trigger\_hook\_args** (*tuple*) – Arguments for the trigger condition

**Yields** Event

**register\_instrument** (*instrument*, *label='instrument'*)

Register an instrument and assign a channel to it. Instruments must have a pull() method which allows to pull data from them at a certain event.

**Parameters**

- **instrument** (*ducktype*) – needs to be configured already and must have a pull() method
- **channel\_name** (*int*) – identify the instrument under this registered channel

**Returns** None

**class** skippylab.daq.**Event** (*use\_datetime=False*)

Bases: `object`

DAQ will return events when triggered.

**timestamp\_it** ()

Give it a timestamp! Time in seconds

**Returns** None

## 1.1.4 skippylab.loggers module

Prepare logging functionality for the module

`skippylab.loggers.get_logger` (*loglevel*, *logfile=None*)

A root logger with a formatted output logging to stdout and a file

**Parameters**

- **loglevel** (*int*) – 10,20,30,... the higher the less logging
- **logfile** (*str*) – write logging to this file as well as stdout

**Returns** logging.logger

## 1.1.5 skippylab.plotting module

Convenient plot functions

`skippylab.plotting.plot_histogram` (*bincenters*, *bincontent*, *fig=None*, *savename='test.png'*, *remove\_empty\_bins=True*)

Plot a histogram returned by TektronixDPO4104B.get\_histogram Use pylab.plot

**Parameters**

- **bincenters** (*np.ndarray*); *bincenters* (*x*) –
- **bincontent** (*np.ndarray*) – bincontent (*y*)

**Keyword Arguments**

- **fig** (*pylab.figure*) – A figure instance
- **savename** (*str*) – where to save the figure (full path)
- **remove\_empty\_bins** (*bool*) – Cut away preceding and trailing zero bins

`skippylab.plotting.plot_waveform(wf_header, wf_data, fig=None, savename=None, use_mv_and_ns=True, color=None)`

Make a plot of a single acquisition

**Parameters**

- **wf\_header** (*dict*) – custom waveform header
- **wf\_data** (*np.ndarray*) – waveform data

**Keyword Arguments**

- **fig** (*pylab.figure*) – A figure instance
- **savename** (*str*) – where to save the figure (full path)
- **use\_mv\_and\_ns** (*bool*) – use mV and ns instead of V and s

**Returns** *pylab.fig*

## 1.1.6 skippylab.tools module

Convenient operations

`skippylab.tools.average_wf(waveforms)`

Get the average waveform

**Parameters** *waveforms* (*iterable of np.ndarrays*) –

**Returns** *np.ndarray*

`skippylab.tools.integrate_wf(waveform, xs, xstep, method=<Mock name='mock.integrate.simps' id='140338442310376'>, impedance=50)`

Integrate a waveform, i.e. a voltage curve. If the desired result shall be indeed a charge, please make sure to give *xs* in seconds and impedance in Ohm accordingly. *xstep* needs to be in seconds as well.

**Parameters**

- **waveform** (*np.ndarray*) – voltage values
- **xs** (*np.ndarray*) – timing values
- **xstep** (*float*) – timing bin size

**Keyword Arguments**

- **method** (*func*) – integration method
- **impedance** (*float*) – needed to calculate actual charge

**Returns** *float*

`skippylab.tools.load_waveform(filename, converter=<function <lambda>>)`

load a waveform from a file

**Parameters** *filename* (*str*) – An existing filename

**Keyword Arguments** **converter** (*func*) – If the data is saved in digitizer levels, use the converter function to convert to Volts

**Returns** *tuple (dict, np.ndarray)*

`skippylab.tools.save_waveform(header, waveform, filename)`

save a waveform together with its header

**Parameters**



- **header** (*dict*) – Some metainformation about the waveform
- **waveform** (*np.ndarray*) – the actual voltage data
- **filename** (*str*) – a filename where the data should be saved

**Returns** None

### 1.1.7 Module contents

Package to read out TektronixDPO4104B oscilloscope



## CHAPTER 2

---

### Indices and tables

---

- [modindex](#)
- [Glossary](#)



### S

skippylab, [13](#)  
skippylab.daq, [10](#)  
skippylab.instruments, [8](#)  
skippylab.instruments.function\_generators,  
    [1](#)  
skippylab.instruments.gpibcontrollers,  
    [1](#)  
skippylab.instruments.oscilloscopes, [2](#)  
skippylab.instruments.powersupplies, [7](#)  
skippylab.loggers, [11](#)  
skippylab.plotting, [11](#)  
skippylab.scpi, [10](#)  
skippylab.scpi.commands, [8](#)  
skippylab.tools, [12](#)



**A**

- AbstractBaseOscilloscope (class in skippy-  
lab.instruments.oscilloscopes), 2
- acquire (skippylab.instruments.oscilloscopes.TektronixDPO4104B  
attribute), 3
- acquire() (skippylab.daq.DAQ method), 10
- acquire\_mode (skippylab.instruments.oscilloscopes.TektronixDPO4104B  
attribute), 3
- acquire\_n\_events() (skippylab.daq.DAQ method), 10
- ACQUIRE\_ONE (skippy-  
lab.instruments.oscilloscopes.AbstractBaseOscilloscope  
attribute), 2
- acquire\_waveform() (skippy-  
lab.instruments.oscilloscopes.AbstractBaseOscilloscope  
method), 2
- acquire\_waveform() (skippy-  
lab.instruments.oscilloscopes.RhodeSchwarzRTO1044  
method), 2
- acquire\_waveform() (skippy-  
lab.instruments.oscilloscopes.TektronixDPO4104B  
method), 3
- acquire\_waveform() (skippy-  
lab.instruments.oscilloscopes.UnknownOscilloscope  
method), 6
- ACQUISITION\_MODE (skippy-  
lab.scpicommands.TektronixDPO4104BCommands  
attribute), 9
- add\_arg() (in module skippylab.scpicommands), 9
- Agilent33220AFunctionGenerator (class in skippy-  
lab.instruments.function\_generators), 1
- APPLY (skippylab.scpicommands.KeysightE3631APowerSupplyCom  
attribute), 8
- average\_waveform() (skippy-  
lab.instruments.oscilloscopes.TektronixDPO4104B  
method), 3
- average\_wf() (in module skippylab.tools), 12
- lab.instruments.oscilloscopes.TektronixDPO4104B  
attribute), 3
- binary\_header\_pattern (skippy-  
lab.instruments.oscilloscopes.TektronixDPO4104B  
attribute), 3

**C**

- CH1 (skippylab.scpicommands.RhodeSchwarzRTO1044Commands  
attribute), 8
- CH1 (skippylab.scpicommands.TektronixDPO4104BCommands  
attribute), 9
- CH2 (skippylab.scpicommands.RhodeSchwarzRTO1044Commands  
attribute), 8
- CH3 (skippylab.scpicommands.TektronixDPO4104BCommands  
attribute), 9
- CH3 (skippylab.scpicommands.RhodeSchwarzRTO1044Commands  
attribute), 8
- CH3 (skippylab.scpicommands.TektronixDPO4104BCommands  
attribute), 9
- CH4 (skippylab.scpicommands.RhodeSchwarzRTO1044Commands  
attribute), 8
- CH4 (skippylab.scpicommands.TektronixDPO4104BCommands  
attribute), 9
- CHANNEL (skippylab.scpicommands.KeysightE3631APowerSupplyCom  
attribute), 8
- clear\_response() (in module skippylab.scpicommands),  
9
- concat() (in module skippylab.scpicommands), 9
- CONTINUOUS\_RUN (skippy-  
lab.instruments.oscilloscopes.AbstractBaseOscilloscope  
attribute), 2
- convert\_waveform() (skippy-  
lab.instruments.oscilloscopes.TektronixDPO4104B  
static method), 3
- CURRENT (skippylab.scpicommands.KeysightE3631APowerSupplyCom  
attribute), 8
- CURVE (skippylab.scpicommands.RhodeSchwarzRTO1044Commands  
attribute), 8

**B**

- binary\_formats (skippy-

## D

DAQ (class in `skippylib.daq`), 10  
 data (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` attribute), 3  
 data\_start (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` attribute), 3  
 data\_stop (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` attribute), 3  
 data\_width (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` attribute), 3

DC (`skippylib.scpicommands.KeysightE3631APowerSupplyCommands` attribute), 8

decode\_ascii\_waveform() (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` static method), 3

decode\_binary\_waveform() (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` method), 3

decode\_header() (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` static method), 4

do\_single\_acquisition() (`skippylib.instruments.oscilloscopes.RhodeSchwarzRTO1044` method), 2

## E

error\_state (`skippylib.instruments.powersupplies.KeysightE3631APowerSupply` attribute), 7

ERROR\_STATEQ (`skippylib.scpicommands.KeysightE3631APowerSupplyCommands` attribute), 8

Event (class in `skippylib.daq`), 11

## F

fill\_buffer() (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` method), 4

## G

get\_header() (in module `skippylib.instruments.oscilloscopes`), 6

get\_logger() (in module `skippylib.loggers`), 11

## H

histbox (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` attribute), 4

histbox\_coordinates() (in module `skippylib.scpicommands`), 9

histend (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` attribute), 4

histogram (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` attribute), 4

histstart (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` attribute), 4

## I

integrate\_wf() (in module `skippylib.tools`), 12

## K

KeysightE3631APowerSupply (class in `skippylib.instruments.powersupplies`), 7

KeysightE3631APowerSupplyCommands (class in `skippylib.scpicommands`), 8

## L

load() (`skippylib.instruments.oscilloscopes.Waveform` method), 6

load\_waveform() (in module `skippylib.tools`), 12

## M

make\_n\_acquisitions() (`skippylib.instruments.oscilloscopes.TektronixDPO4104B` method), 4

MAXTRIALS (`skippylib.instruments.oscilloscopes.AbstractBaseOscilloscope` attribute), 2

MEASURE (`skippylib.scpicommands.KeysightE3631APowerSupplyCom` attribute), 8

measure\_current() (`skippylib.instruments.powersupplies.KeysightE3631APowerSupply` method), 7

## N

N25 (`skippylib.scpicommands.KeysightE3631APowerSupplyCommands` attribute), 8

N\_ACQUISITIONS (`skippylib.scpicommands.TektronixDPO4104BCommands` attribute), 9

N\_ACQUISITIONS (`skippylib.scpicommands.RhodeSchwarzRTO1044Commands` attribute), 8

## O

OFF (`skippylib.scpicommands.KeysightE3631APowerSupplyCommands` attribute), 8

OFF (`skippylib.scpicommands.TektronixDPO4104BCommands` attribute), 9

off() (`skippylib.instruments.powersupplies.KeysightE3631APowerSupply` method), 7

ON (`skippylib.scpicommands.KeysightE3631APowerSupplyCommands` attribute), 8

ON (`skippylib.scpicommands.TektronixDPO4104BCommands` attribute), 9

on() (`skippylib.instruments.powersupplies.KeysightE3631APowerSupply` method), 7

ONE (`skippylib.scpicommands.TektronixDPO4104BCommands` attribute), 9



output (skippylab.instruments.powersupplies.KeysightE3631APowerSupply attribute), 7

OUTPUT (skippylab.scpicommands.KeysightE3631APowerSupplyCommand attribute), 8

**P**

P25 (skippylab.scpicommands.KeysightE3631APowerSupplyCommand attribute), 8

P6 (skippylab.scpicommands.KeysightE3631APowerSupplyCommand attribute), 8

parse\_curve\_data() (in module skippylab.scpicommands), 10

ping() (skippylab.instruments.oscilloscopes.AbstractBaseOscilloscope method), 2

ping() (skippylab.instruments.powersupplies.KeysightE3631APowerSupply method), 7

plot\_histogram() (in module skippylab.plotting), 11

plot\_waveform() (in module skippylab.plotting), 11

prologix\_gpib\_ethernet\_provider() (in module skippylab.instruments.gpibcontrollers), 1

pull() (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 4

**Q**

query() (in module skippylab.scpicommands), 10

**R**

register\_instrument() (skippylab.daq.DAQ method), 11

reopen\_socket() (skippylab.instruments.oscilloscopes.AbstractBaseOscilloscope method), 2

reset\_acquisition\_window() (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 5

RhodeSchwarzRTO1044 (class in skippylab.instruments.oscilloscopes), 2

RhodeSchwarzRTO1044Commands (class in skippylab.scpicommands), 8

RUN (skippylab.scpicommands.RhodeSchwarzRTO1044Commands attribute), 8

run() (skippylab.instruments.oscilloscopes.RhodeSchwarzRTO1044 method), 2

**S**

samplingrate (skippylab.instruments.oscilloscopes.AbstractBaseOscilloscope attribute), 2

samplingrate (skippylab.instruments.oscilloscopes.RhodeSchwarzRTO1044 attribute), 2

samplingrate (skippylab.instruments.oscilloscopes.TektronixDPO4104B attribute), 5

samplingrate() (skippylab.instruments.oscilloscopes.UnknownOscilloscope method), 6

select\_channel() (skippylab.instruments.oscilloscopes.AbstractBaseOscilloscope method), 2

select\_channel() (skippylab.instruments.oscilloscopes.RhodeSchwarzRTO1044 method), 2

select\_channel() (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 5

select\_channel() (skippylab.instruments.oscilloscopes.UnknownOscilloscope method), 6

select\_channel() (skippylab.instruments.powersupplies.KeysightE3631APowerSupply method), 7

set\_acquisition\_window() (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 5

set\_acquisition\_window\_from\_internal\_buffer() (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 5

set\_feature\_acquisition\_window() (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 5

set\_header() (in module skippylab.instruments.oscilloscopes), 6

set\_voltage() (skippylab.instruments.powersupplies.KeysightE3631APowerSupply method), 7

set\_waveform\_encoding() (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 5

setget() (in module skippylab.instruments.oscilloscopes), 6

show\_waveforms() (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 5

SINGLE (skippylab.scpicommands.RhodeSchwarzRTO1044Commands attribute), 8

skippylab (module), 13

skippylab.daq (module), 10

skippylab.instruments (module), 8

skippylab.instruments.function\_generators (module), 1

skippylab.instruments.gpibcontrollers (module), 1

skippylab.instruments.oscilloscopes (module), 2

skippylab.instruments.powersupplies (module), 7

skippylab.loggers (module), 11

skippylab.plotting (module), 11

skippylab.scpicommands (module), 10

skippylab.scpicommands (module), 8

skippylab.tools (module), 12

source (skippylab.instruments.oscilloscopes.TektronixDPO4104B method), 6

attribute), 5

STOP (skippylab.scpicommands.RhodeSchwarzRTO1044Commandsattribute), 9

stop() (skippylab.instruments.oscilloscopes.RhodeSchwarzRTO1044 lab.instruments.oscilloscopes.TektronixDPO4104B method), 3

string\_encoding (skippy- waveform\_enc (skippy- lab.instruments.oscilloscopes.AbstractBaseOscilloscope lab.instruments.oscilloscopes.TektronixDPO4104B attribute), 2

**T**

TektronixDPO4104B (class in skippy- lab.instruments.oscilloscopes), 3

TektronixDPO4104BCommands (class in skippy- lab.scpicommands), 9

time\_bins() (skippylab.instruments.oscilloscopes.Waveform method), 6

time\_binwidth (skippy- waveform\_times (skippy- lab.instruments.oscilloscopes.TektronixDPO4104B lab.instruments.oscilloscopes.TektronixDPO4104B attribute), 5

timestamp\_it() (skippylab.daq.Event method), 11

trigger\_continuous() (skippy- waveform\_bins (skippy- lab.instruments.oscilloscopes.TektronixDPO4104B lab.instruments.oscilloscopes.TektronixDPO4104B attribute), 6

trigger\_frequency\_enabled (skippy- WF (skippylab.scpicommands.TektronixDPO4104BCommands attribute), 6

TRIGGER\_FREQUENCY\_ENABLED (skippy- WF\_BYTEWIDTH (skippy- lab.scpicommands.TektronixDPO4104BCommands attribute), 9

TRIGGER\_FREQUENCYQ (skippy- WF\_HEADER (skippy- lab.scpicommands.RhodeSchwarzRTO1044Commands attribute), 9

trigger\_single() (skippy- WF\_HEADER (skippy- lab.instruments.oscilloscopes.TektronixDPO4104B lab.scpicommands.TektronixDPO4104BCommands attribute), 6

triggerrate (skippylab.instruments.oscilloscopes.RhodeSchwarzRTO1044 attribute), 3

triggerrate (skippylab.instruments.oscilloscopes.TektronixDPO4104B attribute), 6

**U**

UnknownOscilloscope (class in skippy- lab.instruments.oscilloscopes), 6

**V**

VOLT (skippylab.scpicommands.KeysightE3631APowerSupplyCommands attribute), 8

volts() (skippylab.instruments.oscilloscopes.Waveform method), 6

**W**

Waveform (class in skippylab.instruments.oscilloscopes), 6

WAVEFORM (skippylab.scpicommands.RhodeSchwarzRTO1044Comman