Thanks for reading the documentation of the Python package ds1054z. This software makes it easy to talk to your Rigol DS1054Z oscilloscope or any DS1000Z / MSO1000Z series oscilloscope using Python.

ds1054z is free software, published on Github by Philipp Klaus.

It comes with a neat command line tool allowing you to control many functions of your scope such as grabbing waveforms or screenshots, or adjusting settings such as trigger, horizontal, or vertical setup.

Contents:
Installing *ds1054z* is dead simple if you can use pip to install Python packages:

```
pip install ds1054z[savescreen,discovery]
```

The package depends on *python-vxi11* for the communication with the scope. It should automatically get installed along with *ds1054z*.

By specifying *savescreen* and *discovery* in square brackets after the package name, you’re asking pip to install the requirements for those extras as well:

- *savescreen* makes it possible to use the *save-screen* action with the CLI tool. (Pillow will get installed)
- *discovery*: To be able to automatically discover the IP address of the scope on your local network, this extra will install *zeroconf*.

If you don’t have access to pip, the installation might be a bit more tricky. Please let me know how this can be done on your favorite platform and I will add this information here.
This package installs a versatile command line (CLI) tool called ds1054z.

The signature of the command line tool is as follows:

```bash
philipp@lion$ ds1054z --help
usage: ds1054z [-h] [-v] <action> ...

CLI for the DS1054Z scope by Rigol

This tool can be used in very versatile ways. Ask it for --help on the individual actions and it will tell you how to use them.

positional arguments:
  <action> Action to perform on the scope:
  discover Discover and list scopes on your network and exit
  info Print information about your oscilloscope
  cmd Send an SCPI command to the oscilloscope
  save-screen Save an image of the screen
  save-data Save the waveform data to a file
  properties Query properties of the DS1054Z instance
  run Start the oscilloscope data acquisition
  stop Stop the oscilloscope data acquisition
  single Set the oscilloscope to the single trigger mode.
  tforce Generate a trigger signal forcefully.
  shell Start an interactive shell to control your scope.

optional arguments:
  -h, --help show this help message and exit
  -v, --verbose More verbose output
```
Global Options

You can increase the verbosity of the tool by stating `--verbose` before the action argument.

If you want to know what’s going on behind the scenes, and for tracing errors in this software, you might also enable the debugging output by using the undocumented `--debug` parameter. Also put it in front of your action argument.

Saving Screenshots

You can use the tool to save the screen of your scope, for example:

```
ds1054z save-screen --overlay 0.6
```

As a result, a file like this will be saved to your current working directory:

![Screenshot Example]

Zeroconf Device Discovery

Note that no oscilloscope IP address was specified in the last command. This works because the tool performs discovery of DS1000Z devices on the local network. If it finds a single one, it picks that as your device.

If you have multiple oscilloscopes in your network, or want the cli tool to perform your action faster (discovery takes about 1 second upfront), or discovery doesn’t work for you (please file a bug report in that case), then you can just as well specify the scope by its IP address or hostname as a positional parameter to most of the actions:

```
ds1054z save-screen --overlay 0.6 192.168.0.23
```
Exporting Data

You can save the waveform data to a file with the `save-data` command:

```bash
ds1054z save-data --filename samples_{ts}.txt
```
The Class `ds1054z.DS1054Z` is a very easy and convenient way to interact with your oscilloscope programmatically.

First, import the class from the `ds1054z` module:

```python
>>> from ds1054z import DS1054Z
```

Now you're able to instantiate the class providing the host you want to connect to. This can be an IP address or a VISA resources string:

```python
>>> scope = DS1054Z('192.168.0.21')
>>> # or
>>> scope = DS1054Z('TCPIP::192.168.1.104::INSTR')
```

More information on the resources string can be found in the README of the `vxi11` package which `ds1054z.DS1054Z` uses to connect to the scope.

You can then check the identification of the device by accessing its `idn` property:

```python
>>> print(scope.idn)
```

which will print something like `RIGOL TECHNOLOGIES,DS1054Z,DS1ZA116171318,00.04.03`.

To send a command to the oscilloscope, use the `ds1054z.DS1054Z.write()` method. Here we start the scope:

```python
>>> scope.write(':RUN')
```

Note that for those very basic functions there might already be a convenience function present. In this case it's called `ds1054z.DS1054Z.run()`:

```python
>>> scope.run()
```

If you want to read back values from the scope, use the `ds1054z.DS1054Z.query()` method:
>>> scope.query(":ACQuire:SRATe?")
'u'5.000000e+08'

Please note that the answer is given as a (unicode) string here. You still need to convert the value to a float yourself.

>>> float(scope.query(":ACQuire:SRATe?")))  
1000000000.0
This section contains the documentation of the APIs provided by the package `ds1054z`.

Contents:

**The class ds1054z.DS1054Z - Easy communication with your scope**

```python
class ds1054z.DS1054Z(host, *args, **kwargs)
    Bases: vxii1.vxii1.Instrument

    This class represents the oscilloscope.

    Variables
    • product – like 'DS1054Z' (depending on your device)
    • vendor – should be 'RIGOL TECHNOLOGIES'
    • serial – e.g. 'DS1ZA118171631'
    • firmware – e.g. '00.04.03.SP1'

    static decode_ieee_block(ieee_bytes)
    Strips headers (and trailing bytes) from a IEEE binary data block off.

    This is the block format commands like :WAVeform:DATA?, :DISPLAY:DATA?,

    Named after decode_ieee_block() in python-ivi

    display_channel(channel, enable=True)
    Display (enable) or hide (disable) a channel for aquisition and display

    display_only_channel(channel)
    Convenience function to display (enable) a single channel and hide (disable) all others.
```
**static format_si_prefix** *(number, unit=None, as_unicode=True, number_format='{0:.6f}')*

Formats the given number by choosing an appropriate metric prefix and stripping the formatted number of its zero-digits giving a nice human readable form.

If you provide a unit, it will be appended to the resulting string.

Example:

```python
>>> DS1054Z.format_si_prefix(2E-9, unit='s')
'2 ns'
```

**get_channel_measurement** *(channel, item, type='CURRent')*

Measures value on a channel

- **Parameters**
  - `channel` *(int or str)* – The channel name (like CHAN1, ...). Alternatively specify the channel by its number (as integer).
  - `item` *(str)* – Item to measure, can be vmax, vmin, vpp, vtop, vbase, vamp, vavg, vrms, overshoot, preshoot, marea, mparea, period, frequency, rtime, ftime, pwidth, nwidth, pduty, nduty, rdelay, idelay, rphase, fphase, tvmax, tvmin, pslewrate, nslewrate, vupper, vmid, vlower, variance, prms
  - `type` *(str)* – Type of measurement, can be CURRent, MAXimum, MINimum, AVERages, DEViation

**get_channel_offset** *(channel)*

Returns the channel offset in volts.

**get_channel_scale** *(channel)*

Returns the channel scale in volts.

- **Returns** channel scale
  - **Return type** float

**get_probe_ratio** *(channel)*

Returns the probe ratio for a specific channel

**get_waveform_bytes** *(channel, mode='NORMal')*

Get the waveform data for a specific channel as bytes. (In most cases you would want to use the higher level function `get_waveform_samples()` instead.)

This function distinguishes between requests for reading the waveform data currently being displayed on the screen or if you will be reading the internal memory. If you set mode to RAW, the scope will be stopped first and you will get the bytes from internal memory. (Please start it again yourself, if you need to, afterwards.) If you set the mode to MAXimum this function will return the internal memory if the scope is stopped, and the screen memory otherwise.

In case the internal memory will be read, the data request will automatically be split into chunks if it's impossible to read all bytes at once.

- **Parameters**
  - `channel` *(int or str)* – The channel name (like CHAN1, ...). Alternatively specify the channel by its number (as integer).
  - `mode` *(str)* – can be NORMal, MAXimum, or RAW

- **Returns** The waveform data
  - **Return type** bytes
get_waveform_samples(channel, mode='NORMal')

Returns the waveform voltage samples of the specified channel.

The mode argument translates into a call to :WAVEform:MODE setting up how many samples you want to read back. If you set it to normal mode, only the screen content samples will be returned. In raw mode, the whole scope memory will be read out, which can take many seconds depending on the current memory depth.

If you set mode to RAW, the scope will be stopped first. Please start it again yourself, if you need to, afterwards.

If you set mode to NORMal you will always get 1200 samples back. Those 1200 points represent the waveform over the full screen width. This can happen when you stop the acquisition and move the waveform horizontally so that it starts or ends inside the screen area, the missing data points are being set to float('nan') in the list.

Parameters

• channel (int or str) – The channel name (like ‘CHAN1’ or 1).
• mode (str) – can be ‘NORMal’, ‘MAX’, or ‘RAW’

Returns voltage samples

Return type list of float values

query(message, *args, **kwargs)

Write a message to the scope and read back the answer. See vx11.Instrument.ask() for optional parameters.

query_raw(message, *args, **kwargs)

Write a message to the scope and read a (binary) answer. This is the slightly modified version of vx11.Instrument.ask_raw(). It takes a command message string and returns the answer as bytes.

Parameters message (str) – The SCPI command to send to the scope.

Returns Data read from the device

Return type bytes

run()

Start acquisition

set_channel_offset(channel, volts)

Set the (vertical) offset of a specific channel in Volt.

The range of possible offset values depends on the current vertical scale and on the probe ratio. With the probe ratio set to 1x the offset can be set between:

• -100V and +100V (if vertical scale 500mV/div), or
• -2V and +2V (if vertical scale < 500mV/div).

The range scales with the probe ratio. Thus, when the probe ratio is set to 10x, for example, the offset could be set between:

• -1000V and +1000V (if vertical scale 5V/div), or
• -20V and +20V (if vertical scale < 5V/div).

Parameters
- **channel** *(int or str)* – The channel name (like CHAN1, ...). Alternatively specify the channel by its number (as integer).

- **volts** *(float)* – the new vertical scale offset in volts

**set_channel_scale**(channel, volts, use_closest_match=False)

The default steps according to the programming guide:

- 1mV, 2mV, 5mV, 10mV...10V (for a 1x probe),
- 10mV, 20mV, 50mV, 100mV...100V (for a 10x probe).

You can also set the scale to values in between those steps (as with using the fine adjustment mode on the scope).

**Parameters**

- **channel** *(int or str)* – The channel name (like CHAN1, ...). Alternatively specify the channel by its number (as integer).

- **volts** *(float)* – the new value for the vertical channel scaling

- **use_closest_match** *(bool)* – round new scale value to closest match from the default steps

**set_probe_ratio**(channel, ratio)

Set the probe ratio of a specific channel.

The possible ratio values are: 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, and 1000.

**Parameters**

- **channel** *(int or str)* – The channel name (like CHAN1, ...). Alternatively specify the channel by its number (as integer).

- **ratio** *(float)* – Ratio of the probe connected to the channel

**set_waveform_mode**(mode='NORMal')

Changing the waveform mode

- **single**()
  Set the oscilloscope to the single trigger mode.

- **stop**()
  Stop acquisition

- **tforce**()
  Generate a trigger signal forcefully.

**display_data**

The bitmap bytes of the current screen content. This property will be updated every time you access it.

**displayed_channels**

The list of channels currently displayed on the scope. This property will be updated every time you access it.

**idn**

The *IDN?* string of the device. Will be fetched every time you access this property.

**memory_depth**

This maps to the acquisition memory depth the scope is currently set to. In contrast to **memory_depth_curr_waveform**, **memory_depth_internal_currently_shown** and **memory_depth_internal_total**, this property is simply a quite direct access to the :ACQuire:MDEPth command of the oscilloscope.
You can change the memory_depth like this:

```python
>>> scope.memory_depth = 12e6
```

This will set the memory depth to 12M data points. Please note that changing the memory_depth is only possible when the oscilloscope is running. Otherwise, setting this property will raise an exception. In addition, not all values are valid in all cases. This depends on the number of channels currently in use (including the trigger!). Please read the property back after setting it, to check that your desired value was actually acknowledged by the oscilloscope.

When requesting this property, an integer value is returned or the string ‘AUTO’.

This value of this property will be updated every time you access it.

**memory_depth_curr_waveform**

The current memory depth of the oscilloscope. This value is the number of samples to expect when reading the waveform data and depends on the status of the scope (running / stopped).

Needed by `waveform_time_values`.

This property will be updated every time you access it.

**memory_depth_internal_currently_shown**

The number of samples in the raw (=deep) memory of the oscilloscope which are currently being displayed on the screen.

This property will be updated every time you access it.

**memory_depth_internal_total**

The total number of samples in the raw (=deep) memory of the oscilloscope. If it’s running, the scope will be stopped temporarily when accessing this value.

This property will be updated every time you access it.

**timebase_offset**

The timebase offset of the scope in seconds.

You can change the timebase offset by assigning to this property:

```python
>>> scope.timebase_offset = 200e-6
```

The possible values according to the programming manual:

- Screen/2 to 1s or -Screen/2 to 5000s.

**timebase_scale**

The timebase scale of the scope in seconds.

The possible values according to the programming guide:

- Normal mode: 5 ns to 50 s in 1-2-5 steps
- Roll mode: 200 ms to 50 s in 1-2-5 steps

You can change the timebase like this:

```python
>>> scope.timebase_scale = 200E-9
```

The nearest possible value will be set.

**waveform_preamble**

Provides the values returned by the command :WAVEform:PREAMble?. They will be converted to float and int as appropriate.
Those values are essential if you want to convert BYTE data from the scope to voltage readings or if you want to recreate the scope’s display content programmatically.

This property is also accessible via the wrapper property `waveform_preamble_dict` where it returns a `dict` instead of a `tuple`.

This property will be fetched from the scope every time you access it.

**Returns** (fmt, typ, pnts, cnt, xinc, xorig, xref, yinc, yorig, yref)

**Return type** tuple of float and int values

`waveform_preamble_dict` provides a dictionary with 10 entries corresponding to the tuple items of the property `waveform_preamble`.

This property will be fetched from the scope every time you access it.

**Returns** \{'fmt', 'typ', 'pnts', 'cnt', 'xinc', 'xorig', 'xref', 'yinc', 'yorig', 'yref'\}

**Return type** dict

`waveform_time_values` the timestamps that belong to the waveform samples accessed to to be accessed beforehand.

Access this property only after fetching your waveform data, otherwise the values will not be correct.

Will be fetched every time you access this property.

**Returns** sample timestamps (in seconds)

**Return type** list of float

`waveform_time_values_decimal` This is a wrapper for `waveform_time_values`. It returns the time samples as `Decimal` values instead of float which can be convenient for writing with an appropriate precision to a human readable file format.

Access this property only after fetching your waveform data, otherwise the values will not be correct.

Will be fetched every time you access this property.

**Returns** sample timestamps (in seconds)

**Return type** list of `Decimal`

The submodule `ds1054z.discovery - Zeroconf Discovery for Rigol DS1000Z-series scopes`

This submodule depends on the Python package `zeroconf`. Thus,

```python
>>> import ds1054z.discovery
```

raises an `ImportError` in case, the zeroconf package is not installed.

```python
class ds1054z.discovery.DS1000ZServiceInfo (type_, name, address=None, port=None, weight=0, priority=0, properties=None, server=None)
```

Patched version of `zeroconf.ServiceInfo` for DS1000Z devices.

```python
ds1054z.discovery.discover_devices (if_any_return_after=0.8, timeout=2.5)```

Discovers Rigol DS1000Z series oscilloscopes on the local networks.
Parameters

- **if\_any\_return\_after** (*float*) – Return after this amount of time in seconds, if at least one device was discovered.

- **timeout** (*float*) – Return after at most this amount of time in seconds whether devices were discovered or not.

**Returns** The list of discovered devices. Each entry is a dictionary containing a ‘model’ and ‘ip’ entry.

**Return type** list of dict
Chapter 4. API Documentation
Python Module Index

d
  ds1054z, 11
  ds1054z.discovery, 16
Index

D
decode_ieee_block() (ds1054z.DS1054Z static method), 11
discover_devices() (in module ds1054z.discovery), 16
display_channel() (ds1054z.DS1054Z method), 11
display_data (ds1054z.DS1054Z attribute), 14
display_only_channel() (ds1054z.DS1054Z method), 11
displayed_channels (ds1054z.DS1054Z attribute), 14
DS1000ZServiceInfo (class in ds1054z.discovery), 16
DS1054Z (class in ds1054z), 11
ds1054z (module), 11
ds1054z.discovery (module), 16

F
format_si_prefix() (ds1054z.DS1054Z static method), 11

G
get_channel_measurement() (ds1054z.DS1054Z method), 12
get_channel_offset() (ds1054z.DS1054Z method), 12
get_channel_scale() (ds1054z.DS1054Z method), 12
get_probe_ratio() (ds1054z.DS1054Z method), 12
get_waveform_bytes() (ds1054z.DS1054Z method), 12
get_waveform_samples() (ds1054z.DS1054Z method), 12

I
idn (ds1054z.DS1054Z attribute), 14

M
memory_depth (ds1054z.DS1054Z attribute), 14
memory_depth_curr_waveform (ds1054z.DS1054Z attribute), 15
memory_depth_internal_currently_shown (ds1054z.DS1054Z attribute), 15
memory_depth_internal_total (ds1054z.DS1054Z attribute), 15

Q
query() (ds1054z.DS1054Z method), 13

R
run() (ds1054z.DS1054Z method), 13

S
set_channel_offset() (ds1054z.DS1054Z method), 13
set_channel_scale() (ds1054z.DS1054Z method), 14
set_probe_ratio() (ds1054z.DS1054Z method), 14
set_waveform_mode() (ds1054z.DS1054Z method), 14
single() (ds1054z.DS1054Z method), 14
stop() (ds1054z.DS1054Z method), 14

T
tforce() (ds1054z.DS1054Z method), 14
timebase_offset (ds1054z.DS1054Z attribute), 15
timebase_scale (ds1054z.DS1054Z attribute), 15

W
waveform_preamble (ds1054z.DS1054Z attribute), 15
waveform_preamble_dict (ds1054z.DS1054Z attribute), 16
waveform_time_values (ds1054z.DS1054Z attribute), 16
waveform_time_values_decimal (ds1054z.DS1054Z attribute), 16