

---

# NI-SCOPE Python API Documentation

*Release 1.4.8*

NI

Apr 26, 2024



# DOCUMENTATION

<b>1</b>	<b>About</b>	<b>1</b>
1.1	Support Policy . . . . .	1
<b>2</b>	<b>Contributing</b>	<b>3</b>
<b>3</b>	<b>Support / Feedback</b>	<b>5</b>
<b>4</b>	<b>Bugs / Feature Requests</b>	<b>7</b>
4.1	niscscope module . . . . .	7
4.1.1	Installation . . . . .	7
4.1.2	Usage . . . . .	7
4.1.3	API Reference . . . . .	8
4.2	Additional Documentation . . . . .	150
<b>5</b>	<b>License</b>	<b>151</b>
<b>6</b>	<b>Indices and tables</b>	<b>153</b>
	<b>Python Module Index</b>	<b>155</b>
	<b>Index</b>	<b>157</b>



## ABOUT

The **niscopes** module provides a Python API for NI-SCOPE. The code is maintained in the Open Source repository for [nimi-python](#).

### 1.1 Support Policy

**niscopes** supports all the Operating Systems supported by NI-SCOPE.

It follows [Python Software Foundation](#) support policy for different versions of CPython.



## CONTRIBUTING

We welcome contributions! You can clone the project repository, build it, and install it by [following these instructions](#).





## SUPPORT / FEEDBACK

For support specific to the Python API, follow the processs in [Bugs / Feature Requests](#). For support with hardware, the driver runtime or any other questions not specific to the Python API, please visit [NI Community Forums](#).



## BUGS / FEATURE REQUESTS

To report a bug or submit a feature request specific to Python API, please use the [GitHub issues page](#).

Fill in the issue template as completely as possible and we will respond as soon as we can.

### 4.1 niscscope module

#### 4.1.1 Installation

As a prerequisite to using the **niscscope** module, you must install the NI-SCOPE runtime on your system. Visit [ni.com/downloads](#) to download the driver runtime for your devices.

The nimi-python modules (i.e. for **NI-SCOPE**) can be installed with **pip**:

```
$ python -m pip install niscscope~=1.4.8
```

#### 4.1.2 Usage

The following is a basic example of using the **niscscope** module to open a session to a High Speed Digitizer and capture a single record of 1000 points.

```
import niscscope
with niscscope.Session("Dev1") as session:
    session.channels[0].configure_vertical(range=1.0, coupling=niscscope.VerticalCoupling.
    ↪AC)
    session.channels[1].configure_vertical(range=10.0, coupling=niscscope.VerticalCoupling.
    ↪DC)
    session.configure_horizontal_timing(min_sample_rate=500000000, min_num_pts=1000, ref_
    ↪position=50.0, num_records=5, enforce_realtime=True)
    with session.initiate():
        waveforms = session.channels[0,1].fetch(num_records=5)
        for wfm in waveforms:
            print('Channel {}, record {} samples acquired: {:,}\n'.format(wfm.channel, wfm.
            ↪record, len(wfm.samples)))

    # Find all channel 1 records (Note channel name is always a string even if integers.
    ↪used in channel[])
    chan1 = [wfm for wfm in waveforms if wfm.channel == '0']
```

(continues on next page)

(continued from previous page)

```
# Find all record number 3
rec3 = [wfm for wfm in waveforms if wfm.record == 3]
```

If you need faster fetch performance, or to directly interface with `SciPy`, you can use the `fetch_into()` method instead of `fetch()`. See the `fetch_into` example.

Other usage examples can be found on [GitHub](#).

## 4.1.3 API Reference

### Session

```
class niscopes.Session(self, resource_name, id_query=False, reset_device=False, options={}, *,
                        grpc_options=None)
```

Performs the following initialization actions:

- Creates a new IVI instrument driver and optionally sets the initial state of the following session properties: Range Check, Cache, Simulate, Record Value Coercions
- Opens a session to the specified device using the interface and address you specify for the **resourceName**
- Resets the digitizer to a known state if **resetDevice** is set to True
- Queries the instrument ID and verifies that it is valid for this instrument driver if the **IDQuery** is set to True
- Returns an instrument handle that you use to identify the instrument in all subsequent instrument driver method calls

#### Parameters

- **resource\_name** (*str*) –

**Caution:** Traditional NI-DAQ and NI-DAQmx device names are not case-sensitive. However, all IVI names, such as logical names, are case-sensitive. If you use logical names, driver session names, or virtual names in your program, you must make sure that the name you use matches the name in the IVI Configuration Store file exactly, without any variations in the case of the characters.

Specifies the resource name of the device to initialize

For Traditional NI-DAQ devices, the syntax is `DAQ::n`, where *n* is the device number assigned by MAX, as shown in Example 1.

For NI-DAQmx devices, the syntax is just the device name specified in MAX, as shown in Example 2. Typical default names for NI-DAQmx devices in MAX are `Dev1` or `PXI1Slot1`. You can rename an NI-DAQmx device by right-clicking on the name in MAX and entering a new name.

An alternate syntax for NI-DAQmx devices consists of `DAQ::NI-DAQmx device name`, as shown in Example 3. This naming convention allows for the use of an NI-DAQmx device in an application that was originally designed for a Traditional NI-DAQ device. For example,

if the application expects DAQ::1, you can rename the NI-DAQmx device to 1 in MAX and pass in DAQ::1 for the resource name, as shown in Example 4.

If you use the DAQ::*n* syntax and an NI-DAQmx device name already exists with that same name, the NI-DAQmx device is matched first.

You can also pass in the name of an IVI logical name or an IVI virtual name configured with the IVI Configuration utility, as shown in Example 5. A logical name identifies a particular virtual instrument. A virtual name identifies a specific device and specifies the initial settings for the session.

Example	Device Type	Syntax
1	Traditional NI-DAQ device	DAQ::1 (1 = device number)
2	NI-DAQmx device	myDAQmxDevice (myDAQmxDevice = device name)
3	NI-DAQmx device	DAQ::myDAQmxDevice (myDAQmxDevice = device name)
4	NI-DAQmx device	DAQ::2 (2 = device name)
5	IVI logical name or IVI virtual name	myLogicalName (myLogicalName = name)

- **id\_query** (*bool*) – Specify whether to perform an ID query.

When you set this parameter to True, NI-SCOPE verifies that the device you initialize is a type that it supports.

When you set this parameter to False, the method initializes the device without performing an ID query.

#### Defined Values

True—Perform ID query

False—Skip ID query

**Default Value:** True

- **reset\_device** (*bool*) – Specify whether to reset the device during the initialization process.

Default Value: True

#### Defined Values

True (1)—Reset device

False (0)—Do not reset device

---

**Note:** For the NI 5112, repeatedly resetting the device may cause excessive wear on the electromechanical relays. Refer to [NI 5112 Electromechanical Relays](#) for recommended programming practices.

---

- **options** (*dict*) – Specifies the initial value of certain properties for the session. The syntax for **options** is a dictionary of properties with an assigned value. For example:

```
{ 'simulate': False }
```

You do not have to specify a value for all the properties. If you do not specify a value for a property, the default value is used.

Advanced Example: { 'simulate': True, 'driver\_setup': { 'Model': '<model number>', 'BoardType': '<type>' } }

Property	Default
range_check	True
query_instrument_status	False
cache	True
simulate	False
record_value_coersions	False
driver_setup	{ }

- **grpc\_options** (`niscope.GrpcSessionOptions`) – MeasurementLink gRPC session options

## Methods

### abort

`niscope.Session.abort()`

Aborts an acquisition and returns the digitizer to the Idle state. Call this method if the digitizer times out waiting for a trigger.

### acquisition\_status

`niscope.Session.acquisition_status()`

Returns status information about the acquisition to the **status** output parameter.

#### Return type

`niscope.AcquisitionStatus`

#### Returns

Returns whether the acquisition is complete, in progress, or unknown.

#### Defined Values

`COMPLETE`

`IN_PROGRESS`

`STATUS_UNKNOWN`

## add\_waveform\_processing

`niscope.Session.add_waveform_processing(meas_function)`

Adds one measurement to the list of processing steps that are completed before the measurement. The processing is added on a per channel basis, and the processing measurements are completed in the same order they are registered. All measurement library parameters—the properties starting with “**meas\_**”—are cached at the time of registering the processing, and this set of parameters is used during the processing step. The processing measurements are streamed, so the result of the first processing step is used as the input for the next step. The processing is done before any other measurements.

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].add_waveform_processing()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.add_waveform_processing()`

---

### Parameters

**meas\_function** (`niscope.ArrayMeasurement`) – The array measurement to add.

## auto\_setup

`niscope.Session.auto_setup()`

Automatically configures the instrument. When you call this method, the digitizer senses the input signal and automatically configures many of the instrument settings. If a signal is detected on a channel, the driver chooses the smallest available vertical range that is larger than the signal range. For example, if the signal is a 1.2 V<sub>pk-pk</sub> sine wave, and the device supports 1 V and 2 V vertical ranges, the driver will choose the 2 V vertical range for that channel.

If no signal is found on any analog input channel, a warning is returned, and all channels are enabled. A channel is considered to have a signal present if the signal is at least 10% of the smallest vertical range available for that channel.

The following settings are changed:

General	
Acquisition mode	Normal
Reference clock	Internal
Vertical	
Vertical coupling	AC (DC for NI 5621)
Vertical bandwidth	Full
Vertical range	Changed by auto setup
Vertical offset	0 V
Probe attenuation	Unchanged by auto setup
Input impedance	Unchanged by auto setup
Horizontal	
Sample rate	Changed by auto setup
Min record length	Changed by auto setup
Enforce realtime	True
Number of Records	Changed to 1
Triggering	
Trigger type	Edge if signal present, otherwise immediate
Trigger channel	Lowest numbered channel with a signal present
Trigger slope	Positive
Trigger coupling	DC
Reference position	50%
Trigger level	50% of signal on trigger channel
Trigger delay	0
Trigger holdoff	0
Trigger output	None

### `clear_waveform_measurement_stats`

`niscope.Session.clear_waveform_measurement_stats(clearable_measurement_function=niscope.ClearableMeasurementFunction)`

Clears the waveform stats on the channel and measurement you specify. If you want to clear all of the measurements, use [ALL\\_MEASUREMENTS](#) in the `clearableMeasurementFunction` parameter.

Every time a measurement is called, the statistics information is updated, including the min, max, mean, standard deviation, and number of updates. This information is fetched with `niscope.Session._fetch_measurement_stats()`. The multi-acquisition array measurements are also cleared with this method.

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].clear_waveform_measurement_stats()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.clear_waveform_measurement_stats()`

---

#### Parameters

**`clearable_measurement_function`** (`niscope.ClearableMeasurementFunction`) – The scalar measurement or array measurement to clear the stats for.



## clear\_waveform\_processing

`niscope.Session.clear_waveform_processing()`

Clears the list of processing steps assigned to the given channel. The processing is added using the `niscope.Session.add_waveform_processing()` method, where the processing steps are completed in the same order in which they are registered. The processing measurements are streamed, so the result of the first processing step is used as the input for the next step. The processing is also done before any other measurements.

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].clear_waveform_processing()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.clear_waveform_processing()`

---

## close

`niscope.Session.close()`

When you are finished using an instrument driver session, you must call this method to perform the following actions:

- Closes the instrument I/O session.
- Destroys the IVI session and all of its properties.
- Deallocates any memory resources used by the IVI session.

---

**Note:** This method is not needed when using the session context manager

---

## commit

`niscope.Session.commit()`

Commits to hardware all the parameter settings associated with the task. Use this method if you want a parameter change to be immediately reflected in the hardware. This method is not supported for Traditional NI-DAQ (Legacy) devices.

## configure\_chan\_characteristics

`niscope.Session.configure_chan_characteristics(input_impedance, max_input_frequency)`

Configures the properties that control the electrical characteristics of the channel—the input impedance and the bandwidth.

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].configure_chan_characteristics()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.configure_chan_characteristics()`

---

#### Parameters

- **input\_impedance** (*float*) – The input impedance for the channel; NI-SCOPE sets `niscope.Session.input_impedance` to this value.
- **max\_input\_frequency** (*float*) – The bandwidth for the channel; NI-SCOPE sets `niscope.Session.max_input_frequency` to this value. Pass 0 for this value to use the hardware default bandwidth. Pass -1 for this value to achieve full bandwidth.

### configure\_equalization\_filter\_coefficients

`niscope.Session.configure_equalization_filter_coefficients(coefficients)`

Configures the custom coefficients for the equalization FIR filter on the device. This filter is designed to compensate the input signal for artifacts introduced to the signal outside of the digitizer. Because this filter is a generic FIR filter, any coefficients are valid. Coefficient values should be between +1 and -1.

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].configure_equalization_filter_coefficients()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.configure_equalization_filter_coefficients()`

---

#### Parameters

**coefficients** (*list of float*) – The custom coefficients for the equalization FIR filter on the device. These coefficients should be between +1 and -1. You can obtain the number of coefficients from the `:py:attr: `niscope.Session.equalization_num_coefficients`` `<cvi:py:attr:niscope.Session.equalization_num_coefficients.html>` property. The `:py:attr: `niscope.Session.equalization_filter_enabled`` `<cvi:py:attr:niscope.Session.equalization_filter_enabled.html>` property must be set to TRUE to enable the filter.

### configure\_horizontal\_timing

`niscope.Session.configure_horizontal_timing(min_sample_rate, min_num_pts, ref_position, num_records, enforce_realtime)`

Configures the common properties of the horizontal subsystem for a multirecord acquisition in terms of minimum sample rate.

#### Parameters

- **min\_sample\_rate** (*float*) – The sampling rate for the acquisition. Refer to [niscope.Session.min\\_sample\\_rate](#) for more information.
- **min\_num\_pts** (*int*) – The minimum number of points you need in the record for each channel; call `niscopesession.ActualRecordLength()` to obtain the actual record length used.

Valid Values: Greater than 1; limited by available memory

---

**Note:** One or more of the referenced methods are not in the Python API for this driver.

---

- **ref\_position** (*float*) – The position of the Reference Event in the waveform record specified as a percentage.
- **num\_records** (*int*) – The number of records to acquire
- **enforce\_realtime** (*bool*) – Indicates whether the digitizer enforces real-time measurements or allows equivalent-time (RIS) measurements; not all digitizers support RIS—refer to [Features Supported by Device](#) for more information.

Default value: True

#### Defined Values

True—Allow real-time acquisitions only

False—Allow real-time and equivalent-time acquisitions

## configure\_trigger\_digital

```
niscopesession.configure_trigger_digital(trigger_source,  
                                         slope=niscopesession.TriggerSlope.POSITIVE,  
                                         holdoff=hightime.timedelta(seconds=0.0),  
                                         delay=hightime.timedelta(seconds=0.0))
```

Configures the common properties of a digital trigger.

When you initiate an acquisition, the digitizer waits for the start trigger, which is configured through the [niscope.Session.acq\\_arm\\_source](#) (Start Trigger Source) property. The default is immediate. Upon receiving the start trigger the digitizer begins sampling pretrigger points. After the digitizer finishes sampling pretrigger points, the digitizer waits for a reference (stop) trigger that you specify with a method such as this one. Upon receiving the reference trigger the digitizer finishes the acquisition after completing posttrigger sampling. With each Configure Trigger method, you specify configuration parameters such as the trigger source and the amount of trigger delay.

---

**Note:** For multirecord acquisitions, all records after the first record are started by using the Advance Trigger Source. The default is immediate.

You can adjust the amount of pre-trigger and post-trigger samples using the reference position parameter on the [niscope.Session.configure\\_horizontal\\_timing\(\)](#) method. The default is half of the record length.

Some features are not supported by all digitizers. Refer to [Features Supported by Device](#) for more information.

Digital triggering is not supported in RIS mode.

---

### Parameters

- **trigger\_source** (*str*) – Specifies the trigger source. Refer to [\*niscope.Session.trigger\\_source\*](#) for defined values.
- **slope** ([\*niscope.TriggerSlope\*](#)) – Specifies whether you want a rising edge or a falling edge to trigger the digitizer. Refer to [\*niscope.Session.trigger\\_slope\*](#) for more information.
- **holdoff** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The length of time the digitizer waits after detecting a trigger before enabling NI-SCOPE to detect another trigger. Refer to [\*niscope.Session.trigger\\_holdoff\*](#) for more information.
- **delay** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – How long the digitizer waits after receiving the trigger to start acquiring data. Refer to [\*niscope.Session.trigger\\_delay\\_time\*](#) for more information.

### configure\_trigger\_edge

```
niscope.Session.configure_trigger_edge(trigger_source, level, trigger_coupling,  
                                      slope=niscope.TriggerSlope.POSITIVE,  
                                      holdoff=hightime.timedelta(seconds=0.0),  
                                      delay=hightime.timedelta(seconds=0.0))
```

Configures common properties for analog edge triggering.

When you initiate an acquisition, the digitizer waits for the start trigger, which is configured through the [\*niscope.Session.acq\\_arm\\_source\*](#) (Start Trigger Source) property. The default is immediate. Upon receiving the start trigger the digitizer begins sampling pretrigger points. After the digitizer finishes sampling pretrigger points, the digitizer waits for a reference (stop) trigger that you specify with a method such as this one. Upon receiving the reference trigger the digitizer finishes the acquisition after completing posttrigger sampling. With each Configure Trigger method, you specify configuration parameters such as the trigger source and the amount of trigger delay.

---

**Note:** Some features are not supported by all digitizers. Refer to [Features Supported by Device](#) for more information.

---

### Parameters

- **trigger\_source** (*str*) – Specifies the trigger source. Refer to [\*niscope.Session.trigger\\_source\*](#) for defined values.
- **level** (*float*) – The voltage threshold for the trigger. Refer to [\*niscope.Session.trigger\\_level\*](#) for more information.
- **trigger\_coupling** ([\*niscope.TriggerCoupling\*](#)) – Applies coupling and filtering options to the trigger signal. Refer to [\*niscope.Session.trigger\\_coupling\*](#) for more information.
- **slope** ([\*niscope.TriggerSlope\*](#)) – Specifies whether you want a rising edge or a falling edge to trigger the digitizer. Refer to [\*niscope.Session.trigger\\_slope\*](#) for more information.
- **holdoff** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The length of time the digitizer waits after detecting a trigger before

enabling NI-SCOPE to detect another trigger. Refer to [`niscope.Session.trigger\_holdoff`](#) for more information.

- **delay** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – How long the digitizer waits after receiving the trigger to start acquiring data. Refer to [`niscope.Session.trigger\_delay\_time`](#) for more information.

## configure\_trigger\_hysteresis

```
niscope.Session.configure_trigger_hysteresis(trigger_source, level, hysteresis,  
                                             trigger_coupling,  
                                             slope=niscope.TriggerSlope.POSITIVE,  
                                             holdoff=hightime.timedelta(seconds=0.0),  
                                             delay=hightime.timedelta(seconds=0.0))
```

Configures common properties for analog hysteresis triggering. This kind of trigger specifies an additional value, specified in the **hysteresis** parameter, that a signal must pass through before a trigger can occur. This additional value acts as a kind of buffer zone that keeps noise from triggering an acquisition.

When you initiate an acquisition, the digitizer waits for the start trigger, which is configured through the [`niscope.Session.acq\_arm\_source`](#). The default is immediate. Upon receiving the start trigger the digitizer begins sampling pretrigger points. After the digitizer finishes sampling pretrigger points, the digitizer waits for a reference (stop) trigger that you specify with a method such as this one. Upon receiving the reference trigger the digitizer finishes the acquisition after completing post-trigger sampling. With each Configure Trigger method, you specify configuration parameters such as the trigger source and the amount of trigger delay.

---

**Note:** Some features are not supported by all digitizers. Refer to [Features Supported by Device](#) for more information.

---

### Parameters

- **trigger\_source** (*str*) – Specifies the trigger source. Refer to [`niscope.Session.trigger\_source`](#) for defined values.
- **level** (*float*) – The voltage threshold for the trigger. Refer to [`niscope.Session.trigger\_level`](#) for more information.
- **hysteresis** (*float*) – The size of the hysteresis window on either side of the **level** in volts; the digitizer triggers when the trigger signal passes through the hysteresis value you specify with this parameter, has the slope you specify with **slope**, and passes through the **level**. Refer to [`niscope.Session.trigger\_hysteresis`](#) for defined values.
- **trigger\_coupling** ([`niscope.TriggerCoupling`](#)) – Applies coupling and filtering options to the trigger signal. Refer to [`niscope.Session.trigger\_coupling`](#) for more information.
- **slope** ([`niscope.TriggerSlope`](#)) – Specifies whether you want a rising edge or a falling edge to trigger the digitizer. Refer to [`niscope.Session.trigger\_slope`](#) for more information.
- **holdoff** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The length of time the digitizer waits after detecting a trigger before

enabling NI-SCOPE to detect another trigger. Refer to [`niscope.Session.trigger\_holdoff`](#) for more information.

- **delay** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – How long the digitizer waits after receiving the trigger to start acquiring data. Refer to [`niscope.Session.trigger\_delay\_time`](#) for more information.

## configure\_trigger\_immediate

`niscope.Session.configure_trigger_immediate()`

Configures common properties for immediate triggering. Immediate triggering means the digitizer triggers itself.

When you initiate an acquisition, the digitizer waits for a trigger. You specify the type of trigger that the digitizer waits for with a Configure Trigger method, such as [`niscope.Session.configure\_trigger\_immediate\(\)`](#).

## configure\_trigger\_software

`niscope.Session.configure_trigger_software(holdoff=hightime.timedelta(seconds=0.0),  
delay=hightime.timedelta(seconds=0.0))`

Configures common properties for software triggering.

When you initiate an acquisition, the digitizer waits for the start trigger, which is configured through the [`niscope.Session.acq\_arm\_source`](#) (Start Trigger Source) property. The default is immediate. Upon receiving the start trigger the digitizer begins sampling pretrigger points. After the digitizer finishes sampling pretrigger points, the digitizer waits for a reference (stop) trigger that you specify with a method such as this one. Upon receiving the reference trigger the digitizer finishes the acquisition after completing posttrigger sampling. With each Configure Trigger method, you specify configuration parameters such as the trigger source and the amount of trigger delay.

To trigger the acquisition, use [`niscope.Session.send\_software\_trigger\_edge\(\)`](#).

---

**Note:** Some features are not supported by all digitizers. Refer to [Features Supported by Device](#) for more information.

---

### Parameters

- **holdoff** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The length of time the digitizer waits after detecting a trigger before enabling NI-SCOPE to detect another trigger. Refer to [`niscope.Session.trigger\_holdoff`](#) for more information.
- **delay** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – How long the digitizer waits after receiving the trigger to start acquiring data. Refer to [`niscope.Session.trigger\_delay\_time`](#) for more information.

## configure\_trigger\_video

```
niscopesession.Session.configure_trigger_video(trigger_source, signal_format, event, polarity,
                                              trigger_coupling, enable_dc_restore=False,
                                              line_number=1,
                                              holdoff=hightime.timedelta(seconds=0.0),
                                              delay=hightime.timedelta(seconds=0.0))
```

Configures the common properties for video triggering, including the signal format, TV event, line number, polarity, and enable DC restore. A video trigger occurs when the digitizer finds a valid video signal sync.

When you initiate an acquisition, the digitizer waits for the start trigger, which is configured through the `niscopesession.Session.acq_arm_source` (Start Trigger Source) property. The default is immediate. Upon receiving the start trigger the digitizer begins sampling pretrigger points. After the digitizer finishes sampling pretrigger points, the digitizer waits for a reference (stop) trigger that you specify with a method such as this one. Upon receiving the reference trigger the digitizer finishes the acquisition after completing posttrigger sampling. With each Configure Trigger method, you specify configuration parameters such as the trigger source and the amount of trigger delay.

---

**Note:** Some features are not supported by all digitizers. Refer to [Features Supported by Device](#) for more information.

---

### Parameters

- **trigger\_source** (*str*) – Specifies the trigger source. Refer to `niscopesession.Session.trigger_source` for defined values.
- **signal\_format** (`niscopesession.VideoSignalFormat`) – Specifies the type of video signal sync the digitizer should look for. Refer to `niscopesession.tv_trigger_signal_format` for more information.
- **event** (`niscopesession.VideoTriggerEvent`) – Specifies the TV event you want to trigger on. You can trigger on a specific or on the next coming line or field of the signal.
- **polarity** (`niscopesession.VideoPolarity`) – Specifies the polarity of the video signal sync.
- **trigger\_coupling** (`niscopesession.TriggerCoupling`) – Applies coupling and filtering options to the trigger signal. Refer to `niscopesession.Session.trigger_coupling` for more information.
- **enable\_dc\_restore** (*bool*) – Offsets each video line so the clamping level (the portion of the video line between the end of the color burst and the beginning of the active image) is moved to zero volt. Refer to `niscopesession.enable_dc_restore` for defined values.
- **line\_number** (*int*) – Selects the line number to trigger on. The line number range covers an entire frame and is referenced as shown on [Vertical Blanking and Synchronization Signal](#). Refer to `niscopesession.Session.tv_trigger_line_number` for more information.  
Default value: 1
- **holdoff** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The length of time the digitizer waits after detecting a trigger before enabling NI-SCOPE to detect another trigger. Refer to `niscopesession.Session.trigger_holdoff` for more information.



- **delay** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – How long the digitizer waits after receiving the trigger to start acquiring data. Refer to [niscope.Session.trigger\\_delay\\_time](#) for more information.

## configure\_trigger\_window

```
niscope.Session.configure_trigger_window(trigger_source, low_level, high_level,  
                                         window_mode, trigger_coupling,  
                                         holdoff=hightime.timedelta(seconds=0.0),  
                                         delay=hightime.timedelta(seconds=0.0))
```

Configures common properties for analog window triggering. A window trigger occurs when a signal enters or leaves a window you specify with the **high level** or **low level** parameters.

When you initiate an acquisition, the digitizer waits for the start trigger, which is configured through the [niscope.Session.acq\\_arm\\_source](#) (Start Trigger Source) property. The default is immediate. Upon receiving the start trigger the digitizer begins sampling pretrigger points. After the digitizer finishes sampling pretrigger points, the digitizer waits for a reference (stop) trigger that you specify with a method such as this one. Upon receiving the reference trigger the digitizer finishes the acquisition after completing posttrigger sampling. With each Configure Trigger method, you specify configuration parameters such as the trigger source and the amount of trigger delay.

To trigger the acquisition, use [niscope.Session.send\\_software\\_trigger\\_edge\(\)](#).

---

**Note:** Some features are not supported by all digitizers.

---

### Parameters

- **trigger\_source** (*str*) – Specifies the trigger source. Refer to [niscope.Session.trigger\\_source](#) for defined values.
- **low\_level** (*float*) – Passes the voltage threshold you want the digitizer to use for low triggering.
- **high\_level** (*float*) – Passes the voltage threshold you want the digitizer to use for high triggering.
- **window\_mode** ([niscope.TriggerWindowMode](#)) – Specifies whether you want the trigger to occur when the signal enters or leaves a window.
- **trigger\_coupling** ([niscope.TriggerCoupling](#)) – Applies coupling and filtering options to the trigger signal. Refer to [niscope.Session.trigger\\_coupling](#) for more information.
- **holdoff** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The length of time the digitizer waits after detecting a trigger before enabling NI-SCOPE to detect another trigger. Refer to [niscope.Session.trigger\\_holdoff](#) for more information.
- **delay** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – How long the digitizer waits after receiving the trigger to start acquiring data. Refer to [niscope.Session.trigger\\_delay\\_time](#) for more information.



## configure\_vertical

```
niscopesession.configure_vertical(range, coupling, offset=0.0, probe_attenuation=1.0,
                                enabled=True)
```

Configures the most commonly configured properties of the digitizer vertical subsystem, such as the range, offset, coupling, probe attenuation, and the channel.

---

**Tip:** This method can be called on specific channels within your `niscopesession` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].configure_vertical()`

To call the method on all channels, you can call it directly on the `niscopesession`.

Example: `my_session.configure_vertical()`

---

### Parameters

- **range** (*float*) – Specifies the vertical range. Refer to `niscopesession.vertical_range` for more information.
- **coupling** (`niscopesession.VerticalCoupling`) – Specifies how to couple the input signal. Refer to `niscopesession.vertical_coupling` for more information.
- **offset** (*float*) – Specifies the vertical offset. Refer to `niscopesession.vertical_offset` for more information.
- **probe\_attenuation** (*float*) – Specifies the probe attenuation. Refer to `niscopesession.probe_attenuation` for valid values.
- **enabled** (*bool*) – Specifies whether the channel is enabled for acquisition. Refer to `niscopesession.channel_enabled` for more information.

## disable

```
niscopesession.disable()
```

Aborts any current operation, opens data channel relays, and releases RTSI and PFI lines.

## export\_attribute\_configuration\_buffer

```
niscopesession.export_attribute_configuration_buffer()
```

Exports the property configuration of the session to a configuration buffer.

You can export and import session property configurations only between devices with identical model numbers, channel counts, and onboard memory sizes.

This method verifies that the properties you have configured for the session are valid. If the configuration is invalid, NI-SCOPE returns an error.

### Related Topics:

[Properties and Property Methods](#)

[Setting Properties Before Reading Properties](#)

**Return type**`bytes`**Returns**

Specifies the byte array buffer to be populated with the exported property configuration.

**export\_attribute\_configuration\_file**

`niscope.Session.export_attribute_configuration_file(file_path)`

Exports the property configuration of the session to the specified file.

You can export and import session property configurations only between devices with identical model numbers, channel counts, and onboard memory sizes.

This method verifies that the properties you have configured for the session are valid. If the configuration is invalid, NI-SCOPE returns an error.

**Related Topics:**

[Properties and Property Methods](#)

[Setting Properties Before Reading Properties](#)

**Parameters**

**file\_path** (*str*) – Specifies the absolute path to the file to contain the exported property configuration. If you specify an empty or relative path, this method returns an error. **Default file extension:** `.niscopeconfig`

**fetch**

`niscope.Session.fetch(num_samples=None, relative_to=niscope.FetchRelativeTo.PRETRIGGER, offset=0, record_number=0, num_records=None, timeout=hightime.timedelta(seconds=5.0))`

Returns the waveform from a previously initiated acquisition that the digitizer acquires for the specified channel. This method returns scaled voltage waveforms.

This method may return multiple waveforms depending on the number of channels, the acquisition type, and the number of records you specify.

---

**Note:** Some functionality, such as time stamping, is not supported in all digitizers.

---

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].fetch()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.fetch()`

---

**Parameters**

- **num\_samples** (*int*) – The maximum number of samples to fetch for each waveform. If the acquisition finishes with fewer points than requested, some devices return partial data if the acquisition finished, was aborted, or a timeout of 0 was used. If it fails to complete within the timeout period, the method raises.
- **relative\_to** (*niscscope.FetchRelativeTo*) – Position to start fetching within one record.
- **offset** (*int*) – Offset in samples to start fetching data within each record. The offset can be positive or negative.
- **record\_number** (*int*) – Zero-based index of the first record to fetch.
- **num\_records** (*int*) – Number of records to fetch. Use -1 to fetch all configured records.
- **timeout** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The time to wait for data to be acquired; using 0 for this parameter tells NI-SCOPE to fetch whatever is currently available. Using -1 seconds for this parameter implies infinite timeout.

#### Return type

*list of WaveformInfo*

#### Returns

Returns a list of class instances with the following timing and scaling information about each waveform:

- **relative\_initial\_x** (float) the time (in seconds) from the trigger to the first sample in the fetched waveform
- **absolute\_initial\_x** (float) timestamp (in seconds) of the first fetched sample. This timestamp is comparable between records and acquisitions; devices that do not support this parameter use 0 for this output.
- **x\_increment** (float) the time between points in the acquired waveform in seconds
- **channel** (str) channel name this waveform was acquired from
- **record** (int) record number of this waveform
- **gain** (float) the gain factor of the given channel; useful for scaling binary data with the following formula:

$$voltage = binarydata * gainfactor + offset$$

- **offset** (float) the offset factor of the given channel; useful for scaling binary data with the following formula:

$$voltage = binarydata * gainfactor + offset$$

- **samples** (array of float) floating point array of samples. Length will be of the actual samples acquired

## fetch\_array\_measurement

```
niscope.Session.fetch_array_measurement(array_meas_function, meas_wfm_size=None, relative_to=niscope.FetchRelativeTo.PRETRIGGER, offset=0, record_number=0, num_records=None, meas_num_samples=None, timeout=hightime.timedelta(seconds=5.0))
```

Obtains a waveform from the digitizer and returns the specified measurement array. This method may return multiple waveforms depending on the number of channels, the acquisition type, and the number of records you specify.

---

**Note:** Some functionality, such as time stamping, is not supported in all digitizers.

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].fetch_array_measurement()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.fetch_array_measurement()`

---

### Parameters

- **array\_meas\_function** (`niscope.ArrayMeasurement`) – The array measurement to perform.
- **meas\_wfm\_size** (`int`) – The maximum number of samples returned in the measurement waveform array for each waveform measurement. Default Value: `None` (returns all available samples).
- **relative\_to** (`niscope.FetchRelativeTo`) – Position to start fetching within one record.
- **offset** (`int`) – Offset in samples to start fetching data within each record. The offset can be positive or negative.
- **record\_number** (`int`) – Zero-based index of the first record to fetch.
- **num\_records** (`int`) – Number of records to fetch. Use `None` to fetch all configured records.
- **meas\_num\_samples** (`int`) – Number of samples to fetch when performing a measurement. Use `None` to fetch the actual record length.
- **timeout** (`hightime.timedelta`, `datetime.timedelta`, or `float in seconds`) – The time to wait in seconds for data to be acquired; using 0 for this parameter tells NI-SCOPE to fetch whatever is currently available. Using -1 for this parameter implies infinite timeout.

### Return type

`list` of `WaveformInfo`

### Returns

Returns a list of class instances with the following timing and scaling information about each waveform:

- **relativeInitialX**—the time (in seconds) from the trigger to the first sample in the fetched waveform
- **absoluteInitialX**—timestamp (in seconds) of the first fetched sample. This timestamp is comparable between records and acquisitions; devices that do not support this parameter use 0 for this output.
- **xIncrement**—the time between points in the acquired waveform in seconds
- **channel**—channel name this waveform was acquired from
- **record**—record number of this waveform
- **gain**—the gain factor of the given channel; useful for scaling binary data with the following formula:

$$\text{voltage} = \text{binary data} \times \text{gain factor} + \text{offset}$$

- **offset**—the offset factor of the given channel; useful for scaling binary data with the following formula:

$$\text{voltage} = \text{binary data} \times \text{gain factor} + \text{offset}$$

- **samples**—floating point array of samples. Length will be of actual samples acquired.

## fetch\_into

```
niscope.Session.fetch_into(waveform, relative_to=niscope.FetchRelativeTo.PRETRIGGER,
                           offset=0, record_number=0, num_records=None,
                           timeout=hightime.timedelta(seconds=5.0))
```

Returns the waveform from a previously initiated acquisition that the digitizer acquires for the specified channel. This method returns scaled voltage waveforms.

This method may return multiple waveforms depending on the number of channels, the acquisition type, and the number of records you specify.

---

**Note:** Some functionality, such as time stamping, is not supported in all digitizers.

---



---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].fetch()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.fetch()`

---

### Parameters

- **waveform** (`array.array("d")`) – numpy array of the appropriate type and size that should be acquired as a 1D array. Size should be **num\_samples** times number of waveforms. Call `niscope.Session._actual_num_wfms()` to determine the number of waveforms.

Types supported are

- `numpy.float64`
- `numpy.int8`
- `numpy.int16`
- `numpy.int32`

Example:

```
waveform = numpy.ndarray(num_samples * session.actual_num_
    ↪wfms(), dtype=numpy.float64)
wfm_info = session['0,1'].fetch_into(waveform, timeout=5.0)
```

- **relative\_to** (*`niscope.FetchRelativeTo`*) – Position to start fetching within one record.
- **offset** (*`int`*) – Offset in samples to start fetching data within each record. The offset can be positive or negative.
- **record\_number** (*`int`*) – Zero-based index of the first record to fetch.
- **num\_records** (*`int`*) – Number of records to fetch. Use -1 to fetch all configured records.
- **timeout** (*`hightime.timedelta`, `datetime.timedelta`, or float in seconds*) – The time to wait in seconds for data to be acquired; using 0 for this parameter tells NI-SCOPE to fetch whatever is currently available. Using -1 for this parameter implies infinite timeout.

### Return type

*list of WaveformInfo*

### Returns

Returns a list of class instances with the following timing and scaling information about each waveform:

- **relative\_initial\_x** (float) the time (in seconds) from the trigger to the first sample in the fetched waveform
- **absolute\_initial\_x** (float) timestamp (in seconds) of the first fetched sample. This timestamp is comparable between records and acquisitions; devices that do not support this parameter use 0 for this output.
- **x\_increment** (float) the time between points in the acquired waveform in seconds
- **channel** (str) channel name this waveform was acquired from
- **record** (int) record number of this waveform
- **gain** (float) the gain factor of the given channel; useful for scaling binary data with the following formula:

$$voltage = binarydata * gainfactor + offset$$

- **offset** (float) the offset factor of the given channel; useful for scaling binary data with the following formula:

$$voltage = binarydata * gainfactor + offset$$

- **samples** (array of float) floating point array of samples. Length will be of the actual samples acquired

## fetch\_measurement\_stats

```
niscope.Session.fetch_measurement_stats(scalar_meas_function, relative_to=niscope.FetchRelativeTo.PRETRIGGER,
                                         offset=0, record_number=0, num_records=None,
                                         timeout=hightime.timedelta(seconds=5.0))
```

Obtains a waveform measurement and returns the measurement value. This method may return multiple statistical results depending on the number of channels, the acquisition type, and the number of records you specify.

You specify a particular measurement type, such as rise time, frequency, or voltage peak-to-peak. The waveform on which the digitizer calculates the waveform measurement is from an acquisition that you previously initiated. The statistics for the specified measurement method are returned, where the statistics are updated once every acquisition when the specified measurement is fetched by any of the Fetch Measurement methods. If a Fetch Measurement method has not been called, this method fetches the data on which to perform the measurement. The statistics are cleared by calling `niscope.Session.clear_waveform_measurement_stats()`.

Many of the measurements use the low, mid, and high reference levels. You configure the low, mid, and high references with `niscope.Session.meas_chan_low_ref_level`, `niscope.Session.meas_chan_mid_ref_level`, and `niscope.Session.meas_chan_high_ref_level` to set each channel differently.

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].fetch_measurement_stats()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.fetch_measurement_stats()`

---

### Parameters

- **scalar\_meas\_function** (`niscope.ScalarMeasurement`) – The scalar measurement to be performed on each fetched waveform.
- **relative\_to** (`niscope.FetchRelativeTo`) – Position to start fetching within one record.
- **offset** (`int`) – Offset in samples to start fetching data within each record. The offset can be positive or negative.
- **record\_number** (`int`) – Zero-based index of the first record to fetch.
- **num\_records** (`int`) – Number of records to fetch. Use `None` to fetch all configured records.

- **timeout** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The time to wait in seconds for data to be acquired; using 0 for this parameter tells NI-SCOPE to fetch whatever is currently available. Using -1 for this parameter implies infinite timeout.

**Return type**

*list of MeasurementStats*

**Returns**

Returns a list of class instances with the following measurement statistics about the specified measurement:

- **result** (float): the resulting measurement
- **mean** (float): the mean scalar value, which is obtained by

averaging each `fetch_measurement_stats` call - **stdev** (float): the standard deviations of the most recent **numInStats** measurements - **min\_val** (float): the smallest scalar value acquired (the minimum of the **numInStats** measurements) - **max\_val** (float): the largest scalar value acquired (the maximum of the **numInStats** measurements) - **num\_in\_stats** (int): the number of times `fetch_measurement_stats` has been called - **channel** (str): channel name this result was acquired from - **record** (int): record number of this result

## `get_channel_names`

`niscope.Session.get_channel_names(indices)`

Returns a list of channel names for given channel indices.

**Parameters**

**indices** (*basic sequence types, str, or int*) – Index list for the channels in the session. Valid values are from zero to the total number of channels in the session minus one. The index string can be one of the following formats:

- A comma-separated list—for example, “0,2,3,1”
- A range using a hyphen—for example, “0-3”
- A range using a colon—for example, “0:3 “

You can combine comma-separated lists and ranges that use a hyphen or colon. Both out-of-order and repeated indices are supported (“2,3,0”, “1,2,2,3”). White space characters, including spaces, tabs, feeds, and carriage returns, are allowed between characters. Ranges can be incrementing or decrementing.

**Return type**

*list of str*

**Returns**

The channel name(s) at the specified indices.



## get\_equalization\_filter\_coefficients

`niscope.Session.get_equalization_filter_coefficients()`

Retrieves the custom coefficients for the equalization FIR filter on the device. This filter is designed to compensate the input signal for artifacts introduced to the signal outside of the digitizer. Because this filter is a generic FIR filter, any coefficients are valid. Coefficient values should be between +1 and -1.

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].get_equalization_filter_coefficients()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.get_equalization_filter_coefficients()`

---

## get\_ext\_cal\_last\_date\_and\_time

`niscope.Session.get_ext_cal_last_date_and_time()`

Returns the date and time of the last external calibration performed.

**Return type**

`hightime.timedelta`, `datetime.timedelta`, or float in seconds

**Returns**

Indicates the **date** of the last calibration. A `hightime.datetime` object is returned, but only contains resolution to the day.

## get\_ext\_cal\_last\_temp

`niscope.Session.get_ext_cal_last_temp()`

Returns the onboard temperature, in degrees Celsius, of an oscilloscope at the time of the last successful external calibration. The temperature returned by this node is an onboard temperature read from a sensor on the surface of the oscilloscope. This temperature should not be confused with the environmental temperature of the oscilloscope surroundings. During operation, the onboard temperature is normally higher than the environmental temperature. Temperature-sensitive parameters are calibrated during self-calibration. Therefore, the self-calibration temperature is usually more important to read than the external calibration temperature.

**Return type**

`float`

**Returns**

Returns the **temperature** in degrees Celsius during the last calibration.

### `get_self_cal_last_date_and_time`

`niscope.Session.get_self_cal_last_date_and_time()`

Returns the date and time of the last self calibration performed.

**Return type**

`hightime.timedelta`, `datetime.datetime`, or float in seconds

**Returns**

Indicates the **date** of the last calibration. A `hightime.datetime` object is returned, but only contains resolution to the day.

### `get_self_cal_last_temp`

`niscope.Session.get_self_cal_last_temp()`

Returns the onboard temperature, in degrees Celsius, of an oscilloscope at the time of the last successful self calibration. The temperature returned by this node is an onboard temperature read from a sensor on the surface of the oscilloscope. This temperature should not be confused with the environmental temperature of the oscilloscope surroundings. During operation, the onboard temperature is normally higher than the environmental temperature. Temperature-sensitive parameters are calibrated during self-calibration. Therefore, the self-calibration temperature is usually more important to read than the external calibration temperature.

**Return type**

`float`

**Returns**

Returns the **temperature** in degrees Celsius during the last calibration.

### `import_attribute_configuration_buffer`

`niscope.Session.import_attribute_configuration_buffer(configuration)`

Imports a property configuration to the session from the specified configuration buffer.

You can export and import session property configurations only between devices with identical model numbers, channel counts, and onboard memory sizes.

**Related Topics:**

[Properties and Property Methods](#)

[Setting Properties Before Reading Properties](#)

---

**Note:** You cannot call this method while the session is in a running state, such as while acquiring a signal.

---

**Parameters**

**configuration** (*bytes*) – Specifies the byte array buffer that contains the property configuration to import.

## import\_attribute\_configuration\_file

`niscope.Session.import_attribute_configuration_file(file_path)`

Imports a property configuration to the session from the specified file.

You can export and import session property configurations only between devices with identical model numbers, channel counts, and onboard memory sizes.

### Related Topics:

[Properties and Property Methods](#)

[Setting Properties Before Reading Properties](#)

---

**Note:** You cannot call this method while the session is in a running state, such as while acquiring a signal.

---

### Parameters

**file\_path** (*str*) – Specifies the absolute path to the file containing the property configuration to import. If you specify an empty or relative path, this method returns an error. **Default File Extension:** `.niscopeconfig`

## initiate

`niscope.Session.initiate()`

Initiates a waveform acquisition.

After calling this method, the digitizer leaves the Idle state and waits for a trigger. The digitizer acquires a waveform for each channel you enable with `niscope.Session.configure_vertical()`.

---

**Note:** This method will return a Python context manager that will initiate on entering and abort on exit.

---

## lock

`niscope.Session.lock()`

Obtains a multithread lock on the device session. Before doing so, the software waits until all other execution threads release their locks on the device session.

Other threads may have obtained a lock on this session for the following reasons:

- The application called the `niscope.Session.lock()` method.
- A call to NI-SCOPE locked the session.
- After a call to the `niscope.Session.lock()` method returns successfully, no other threads can access the device session until you call the `niscope.Session.unlock()` method or exit out of the with block when using lock context manager.
- Use the `niscope.Session.lock()` method and the `niscope.Session.unlock()` method around a sequence of calls to instrument driver methods if you require that the device retain its settings through the end of the sequence.

You can safely make nested calls to the `niscope.Session.lock()` method within the same thread. To completely unlock the session, you must balance each call to the `niscope.Session.lock()` method with a call to the `niscope.Session.unlock()` method.

One method for ensuring there are the same number of unlock method calls as there is lock calls is to use lock as a context manager

```
with niscope.Session('dev1') as session:
    with session.lock():
        # Calls to session within a single lock context
```

The first *with* block ensures the session is closed regardless of any exceptions raised

The second *with* block ensures that unlock is called regardless of any exceptions raised

#### Return type

context manager

#### Returns

When used in a *with* statement, `niscope.Session.lock()` acts as a context manager and unlock will be called when the *with* block is exited

### probe\_compensation\_signal\_start

`niscope.Session.probe_compensation_signal_start()`

Starts the 1 kHz square wave output on PFI 1 for probe compensation.

### probe\_compensation\_signal\_stop

`niscope.Session.probe_compensation_signal_stop()`

Stops the 1 kHz square wave output on PFI 1 for probe compensation.

### read

`niscope.Session.read(num_samples=None, relative_to=niscope.FetchRelativeTo.PRETRIGGER, offset=0, record_number=0, num_records=None, timeout=hightime.timedelta(seconds=5.0))`

Initiates an acquisition, waits for it to complete, and retrieves the data. The process is similar to calling `niscope.Session._initiate_acquisition()`, `niscope.Session.acquisition_status()`, and `niscope.Session.fetch()`. The only difference is that with `niscope.Session.read()`, you enable all channels specified with `channelList` before the acquisition; in the other method, you enable the channels with `niscope.Session.configure_vertical()`.

This method may return multiple waveforms depending on the number of channels, the acquisition type, and the number of records you specify.

---

**Note:** Some functionality, such as time stamping, is not supported in all digitizers.

---

---

**Tip:** This method can be called on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].read()`

To call the method on all channels, you can call it directly on the `niscope.Session`.

Example: `my_session.read()`

---

### Parameters

- **num\_samples** (*int*) – The maximum number of samples to fetch for each waveform. If the acquisition finishes with fewer points than requested, some devices return partial data if the acquisition finished, was aborted, or a timeout of 0 was used. If it fails to complete within the timeout period, the method raises.
- **relative\_to** (`niscope.FetchRelativeTo`) – Position to start fetching within one record.
- **offset** (*int*) – Offset in samples to start fetching data within each record. The offset can be positive or negative.
- **record\_number** (*int*) – Zero-based index of the first record to fetch.
- **num\_records** (*int*) – Number of records to fetch. Use -1 to fetch all configured records.
- **timeout** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The time to wait for data to be acquired; using 0 for this parameter tells NI-SCOPE to fetch whatever is currently available. Using -1 seconds for this parameter implies infinite timeout.

### Return type

*list of WaveformInfo*

### Returns

Returns a list of class instances with the following timing and scaling information about each waveform:

- **relative\_initial\_x** (float) the time (in seconds) from the trigger to the first sample in the fetched waveform
- **absolute\_initial\_x** (float) timestamp (in seconds) of the first fetched sample. This timestamp is comparable between records and acquisitions; devices that do not support this parameter use 0 for this output.
- **x\_increment** (float) the time between points in the acquired waveform in seconds
- **channel** (str) channel name this waveform was acquired from
- **record** (int) record number of this waveform
- **gain** (float) the gain factor of the given channel; useful for scaling binary data with the following formula:

$$voltage = binarydata * gainfactor + offset$$

- **offset** (float) the offset factor of the given channel; useful for scaling binary data with the following formula:

$$voltage = binarydata * gainfactor + offset$$

- **samples** (array of float) floating point array of samples. Length will be of the actual samples acquired

## reset

`niscope.Session.reset()`

Stops the acquisition, releases routes, and all session properties are reset to their [default states](#).

## reset\_device

`niscope.Session.reset_device()`

Performs a hard reset of the device. Acquisition stops, all routes are released, RTSI and PFI lines are tristated, hardware is configured to its default state, and all session properties are reset to their default state.

- [Thermal Shutdown](#)

## reset\_with\_defaults

`niscope.Session.reset_with_defaults()`

Performs a software reset of the device, returning it to the default state and applying any initial default settings from the IVI Configuration Store.

## self\_cal

`niscope.Session.self_cal(option=niscope.Option.SELF_CALIBRATE_ALL_CHANNELS)`

Self-calibrates most NI digitizers, including all SMC-based devices and most Traditional NI-DAQ (Legacy) devices. To verify that your digitizer supports self-calibration, refer to [Features Supported by Device](#).

For SMC-based digitizers, if the self-calibration is performed successfully in a regular session, the calibration constants are immediately stored in the self-calibration area of the EEPROM. If the self-calibration is performed in an external calibration session, the calibration constants take effect immediately for the duration of the session. However, they are not stored in the EEPROM until `niscope.Session.CalEnd()` is called with **action** set to `NISCOPE_VAL_ACTION_STORE` and no errors occur.

---

**Note:** One or more of the referenced methods are not in the Python API for this driver.

---



---

**Note:** One or more of the referenced values are not in the Python API for this driver. Enums that only define values, or represent True/False, have been removed.

---

---

**Tip:** This method can be called on specific channels within your *niscope.Session* instance. Use Python index notation on the repeated capabilities container channels to specify a subset, and then call this method on the result.

Example: `my_session.channels[ ... ].self_cal()`

To call the method on all channels, you can call it directly on the *niscope.Session*.

Example: `my_session.self_cal()`

---

#### Parameters

**option** (*niscope.Option*) – The calibration option. Use `VI_NULL` for a normal self-calibration operation or `NISCOPE_VAL_CAL_RESTORE_EXTERNAL_CALIBRATION` to restore the previous calibration.

---

**Note:** One or more of the referenced values are not in the Python API for this driver. Enums that only define values, or represent True/False, have been removed.

---

## self\_test

`niscope.Session.self_test()`

Runs the instrument self-test routine and returns the test result(s). Refer to the device-specific help topics for an explanation of the message contents.

Raises *SelfTestError* on self test failure. Properties on exception object:

- code - failure code from driver
- message - status message from driver

Self-Test Code	Description
0	Passed self-test
1	Self-test failed

## send\_software\_trigger\_edge

`niscope.Session.send_software_trigger_edge(which_trigger)`

Sends the selected trigger to the digitizer. Call this method if you called *niscope.Session.configure\_trigger\_software()* when you want the Reference trigger to occur. You can also call this method to override a misused edge, digital, or hysteresis trigger. If you have configured *niscope.Session.acq\_arm\_source*, *niscope.Session.arm\_ref\_trig\_src*, or *niscope.Session.adv\_trig\_src*, call this method when you want to send the corresponding trigger to the digitizer.

#### Parameters

**which\_trigger** (*niscope.WhichTrigger*) – Specifies the type of trigger to send to the digitizer.

#### Defined Values

*START* (0L)  
*ARM\_REFERENCE* (1L)  
*REFERENCE* (2L)  
*ADVANCE* (3L)

## unlock

`niscope.Session.unlock()`

Releases a lock that you acquired on an device session using `niscope.Session.lock()`. Refer to `niscope.Session.unlock()` for additional information on session locks.

## Properties

### absolute\_sample\_clock\_offset

`niscope.Session.absolute_sample_clock_offset`

Gets or sets the absolute time offset of the sample clock relative to the reference clock in terms of seconds.

---

**Note:** Configures the sample clock relationship with respect to the reference clock. This parameter is factored into NI-TClk adjustments and is typically used to improve the repeatability of NI-TClk Synchronization. When this parameter is read, the currently programmed value is returned. The range of the absolute sample clock offset is [-.5 sample clock periods, .5 sample clock periods]. The default absolute sample clock offset is 0s.

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocking:Advanced:Absolute Sample Clock Offset**
  - C Attribute: **NISCOPE\_ATTR\_ABSOLUTE\_SAMPLE\_CLOCK\_OFFSET**
-



## acquisition\_start\_time

### `niscope.Session.acquisition_start_time`

Specifies the length of time from the trigger event to the first point in the waveform record in seconds. If the value is positive, the first point in the waveform record occurs after the trigger event (same as specifying `niscope.Session.trigger_delay_time`). If the value is negative, the first point in the waveform record occurs before the trigger event (same as specifying `niscope.Session.horz_record_ref_position`).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Advanced:Acquisition Start Time**
  - C Attribute: **NISCOPE\_ATTR\_ACQUISITION\_START\_TIME**
- 

## acquisition\_type

### `niscope.Session.acquisition_type`

Specifies how the digitizer acquires data and fills the waveform record.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.AcquisitionType
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Acquisition:Acquisition Type**
  - C Attribute: **NISCOPE\_ATTR\_ACQUISITION\_TYPE**
-

## acq\_arm\_source

### `niscope.Session.acq_arm_source`

Specifies the source the digitizer monitors for a start (acquisition arm) trigger. When the start trigger is received, the digitizer begins acquiring pretrigger samples. Valid Values: NISCOPE\_VAL\_IMMEDIATE ('VAL\_IMMEDIATE') - Triggers immediately NISCOPE\_VAL\_RTISI\_0 ('VAL\_RTISI\_0') - RTISI 0 NISCOPE\_VAL\_RTISI\_1 ('VAL\_RTISI\_1') - RTISI 1 NISCOPE\_VAL\_RTISI\_2 ('VAL\_RTISI\_2') - RTISI 2 NISCOPE\_VAL\_RTISI\_3 ('VAL\_RTISI\_3') - RTISI 3 NISCOPE\_VAL\_RTISI\_4 ('VAL\_RTISI\_4') - RTISI 4 NISCOPE\_VAL\_RTISI\_5 ('VAL\_RTISI\_5') - RTISI 5 NISCOPE\_VAL\_RTISI\_6 ('VAL\_RTISI\_6') - RTISI 6 NISCOPE\_VAL\_PFI\_0 ('VAL\_PFI\_0') - PFI 0 NISCOPE\_VAL\_PFI\_1 ('VAL\_PFI\_1') - PFI 1 NISCOPE\_VAL\_PFI\_2 ('VAL\_PFI\_2') - PFI 2 NISCOPE\_VAL\_PXI\_STAR ('VAL\_PXI\_STAR') - PXI Star Trigger

---

**Note:** One or more of the referenced values are not in the Python API for this driver. Enums that only define values, or represent True/False, have been removed.

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Start Trigger (Acq. Arm):Source**
  - C Attribute: **NISCOPE\_ATTR\_ACQ\_ARM\_SOURCE**
- 

## advance\_trigger\_terminal\_name

### `niscope.Session.advance_trigger_terminal_name`

Returns the fully qualified name for the Advance Trigger terminal. You can use this terminal as the source for another trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Advance Trigger:Terminal Name**
  - C Attribute: **NISCOPE\_ATTR\_ADVANCE\_TRIGGER\_TERMINAL\_NAME**
-

## adv\_trig\_src

### `niscope.Session.adv_trig_src`

Specifies the source the digitizer monitors for an advance trigger. When the advance trigger is received, the digitizer begins acquiring pretrigger samples.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Advance Trigger:Source**
  - C Attribute: **NISCOPE\_ATTR\_ADV\_TRIG\_SRC**
- 

## allow\_more\_records\_than\_memory

### `niscope.Session.allow_more_records_than_memory`

Indicates whether more records can be configured with `niscope.Session.configure_horizontal_timing()` than fit in the onboard memory. If this property is set to True, it is necessary to fetch records while the acquisition is in progress. Eventually, some of the records will be overwritten. An error is returned from the fetch method if you attempt to fetch a record that has been overwritten.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Enable Records > Memory**
  - C Attribute: **NISCOPE\_ATTR\_ALLOW\_MORE\_RECORDS\_THAN\_MEMORY**
-

## arm\_ref\_trig\_src

`niscope.Session.arm_ref_trig_src`

Specifies the source the digitizer monitors for an arm reference trigger. When the arm reference trigger is received, the digitizer begins looking for a reference (stop) trigger from the user-configured trigger source.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Arm Reference Trigger:Source**
  - C Attribute: **NISCOPE\_ATTR\_ARM\_REF\_TRIG\_SRC**
- 

## backlog

`niscope.Session.backlog`

Returns the number of samples (*`niscope.Session.points_done`*) that have been acquired but not fetched for the record specified by `niscope.Session.fetch_record_number`.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Fetch:Fetch Backlog**
  - C Attribute: **NISCOPE\_ATTR\_BACKLOG**
-

## bandpass\_filter\_enabled

### `niscope.Session.bandpass_filter_enabled`

Enables the bandpass filter on the specified channel. The default value is FALSE.

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].bandpass_filter_enabled`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.bandpass_filter_enabled`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Advanced:Bandpass Filter Enabled**
- C Attribute: **NISCOPE\_ATTR\_BANDPASS\_FILTER\_ENABLED**

## binary\_sample\_width

### `niscope.Session.binary_sample_width`

Indicates the bit width of the binary data in the acquired waveform. Useful for determining which Binary Fetch method to use. Compare to `niscope.Session.resolution`. To configure the device to store samples with a lower resolution than the native, set this property to the desired binary width. This can be useful for streaming at faster speeds at the cost of resolution. The least significant bits will be lost with this configuration. Valid Values: 8, 16, 32

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Acquisition:Binary Sample Width**
- C Attribute: **NISCOPE\_ATTR\_BINARY\_SAMPLE\_WIDTH**

## cable\_sense\_mode

### `niscope.Session.cable_sense_mode`

Specifies whether and how the oscilloscope is configured to generate a CableSense signal on the specified channels when the `niscope.Session.CableSenseSignalStart()` method is called.

#### Device-Specific Behavior:

##### PXIe-5160/5162

- The value of this property must be identical across all channels whose input impedance is set to 50 ohms.
- If this property is set to a value other than *DISABLED* for any channel(s), the input impedance of all channels for which this property is set to *DISABLED* must be set to 1 M Ohm.

Supported Devices
PXIe-5110
PXIe-5111
PXIe-5113
PXIe-5160
PXIe-5162

---

**Note:** the input impedance of the channel(s) to convey the CableSense signal must be set to 50 ohms.

---

---

**Note:** One or more of the referenced methods are not in the Python API for this driver.

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.CableSenseMode</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_CABLE\_SENSE\_MODE**
-

## cable\_sense\_signal\_enable

`niscope.Session.cable_sense_signal_enable`

TBD

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_CABLE\_SENSE\_SIGNAL\_ENABLE**

## cable\_sense\_voltage

`niscope.Session.cable_sense_voltage`

Returns the voltage of the CableSense signal that is written to the EEPROM of the oscilloscope during factory calibration.

Supported Devices
PXIe-5110
PXIe-5111
PXIe-5113
PXIe-5160
PXIe-5162

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_CABLE\_SENSE\_VOLTAGE**

## channel\_count

`niscope.Session.channel_count`

Indicates the number of channels that the specific instrument driver supports. For channel-based properties, the IVI engine maintains a separate cache value for each channel.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Driver Capabilities:Channel Count**
  - C Attribute: **NISCOPE\_ATTR\_CHANNEL\_COUNT**
- 

## channel\_enabled

`niscope.Session.channel_enabled`

Specifies whether the digitizer acquires a waveform for the channel. Valid Values: True (1) - Acquire data on this channel False (0) - Don't acquire data on this channel

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].channel_enabled`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.channel_enabled`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Channel Enabled**
  - C Attribute: **NISCOPE\_ATTR\_CHANNEL\_ENABLED**
-



## channel\_terminal\_configuration

`niscope.Session.channel_terminal_configuration`

Specifies the terminal configuration for the channel.

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].channel_terminal_configuration`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.channel_terminal_configuration`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.TerminalConfiguration</code>
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Channel Terminal Configuration**
  - C Attribute: **NISCOPE\_ATTR\_CHANNEL\_TERMINAL\_CONFIGURATION**
- 

## data\_transfer\_block\_size

`niscope.Session.data_transfer_block_size`

Specifies the maximum number of samples to transfer at one time from the device to host memory. Increasing this number should result in better fetching performance because the driver does not need to restart the transfers as often. However, increasing this number may also increase the amount of page-locked memory required from the system.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>int</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Fetch:Data Transfer Block Size**
  - C Attribute: **NISCOPE\_ATTR\_DATA\_TRANSFER\_BLOCK\_SIZE**
-

### `data_transfer_maximum_bandwidth`

`niscope.Session.data_transfer_maximum_bandwidth`

This property specifies the maximum bandwidth that the device is allowed to consume.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Fetch:Advanced:Maximum Bandwidth**
  - C Attribute: **NISCOPE\_ATTR\_DATA\_TRANSFER\_MAXIMUM\_BANDWIDTH**
- 

### `data_transfer_preferred_packet_size`

`niscope.Session.data_transfer_preferred_packet_size`

This property specifies the size of (read request|memory write) data payload. Due to alignment of the data buffers, the hardware may not always generate a packet of this size.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Fetch:Advanced:Preferred Packet Size**
  - C Attribute: **NISCOPE\_ATTR\_DATA\_TRANSFER\_PREFERRED\_PACKET\_SIZE**
- 

### `device_temperature`

`niscope.Session.device_temperature`

Returns the temperature of the device in degrees Celsius from the onboard sensor.

---

**Tip:** This property can be set/get on specific instruments within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container instruments to specify a subset.

Example: `my_session.instruments[ ... ].device_temperature`

To set/get on all instruments, you can call the property directly on the `niscope.Session`.

Example: `my_session.device_temperature`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only
Repeated Capabilities	instruments

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Device:Temperature**
  - C Attribute: **NISCOPE\_ATTR\_DEVICE\_TEMPERATURE**
- 

## enabled\_channels

### `niscope.Session.enabled_channels`

Returns a comma-separated list of the channels enabled for the session in ascending order.

If no channels are enabled, this property returns an empty string, `""`. If all channels are enabled, this property enumerates all of the channels.

Because this property returns channels in ascending order, but the order in which you specify channels for the input is important, the value of this property may not necessarily reflect the order in which NI-SCOPE performs certain actions.

Refer to Channel String Syntax in the NI High-Speed Digitizers Help for more information on the effects of channel order in NI-SCOPE.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_ENABLED\_CHANNELS**
-

## enable\_dc\_restore

### `niscope.Session.enable_dc_restore`

Restores the video-triggered data retrieved by the digitizer to the video signal's zero reference point.  
Valid Values: True - Enable DC restore False - Disable DC restore

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Video:Enable DC Restore**
  - C Attribute: **NISCOPE\_ATTR\_ENABLE\_DC\_RESTORE**
- 

## enable\_time\_interleaved\_sampling

### `niscope.Session.enable_time_interleaved_sampling`

Specifies whether the digitizer acquires the waveform using multiple ADCs for the channel enabling a higher maximum real-time sampling rate. Valid Values: True (1) - Use multiple interleaved ADCs on this channel False (0) - Use only this channel's ADC to acquire data for this channel

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].enable_time_interleaved_sampling`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.enable_time_interleaved_sampling`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Enable Time Interleaved Sampling**
  - C Attribute: **NISCOPE\_ATTR\_ENABLE\_TIME\_INTERLEAVED\_SAMPLING**
-

## end\_of\_acquisition\_event\_output\_terminal

### `niscope.Session.end_of_acquisition_event_output_terminal`

Specifies the destination for the End of Acquisition Event. When this event is asserted, the digitizer has completed sampling for all records. Consult your device documentation for a specific list of valid destinations.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:End of Acquisition:Output Terminal**
  - C Attribute: **NISCOPE\_ATTR\_END\_OF\_ACQUISITION\_EVENT\_OUTPUT\_TERMINAL**
- 

## end\_of\_acquisition\_event\_terminal\_name

### `niscope.Session.end_of_acquisition_event_terminal_name`

Returns the fully qualified name for the End of Acquisition Event terminal. You can use this terminal as the source for a trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:End of Acquisition:Terminal Name**
  - C Attribute: **NISCOPE\_ATTR\_END\_OF\_ACQUISITION\_EVENT\_TERMINAL\_NAME**
-

## end\_of\_record\_event\_output\_terminal

`niscope.Session.end_of_record_event_output_terminal`

Specifies the destination for the End of Record Event. When this event is asserted, the digitizer has completed sampling for the current record. Consult your device documentation for a specific list of valid destinations.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:End of Record:Output Terminal**
  - C Attribute: **NISCOPE\_ATTR\_END\_OF\_RECORD\_EVENT\_OUTPUT\_TERMINAL**
- 

## end\_of\_record\_event\_terminal\_name

`niscope.Session.end_of_record_event_terminal_name`

Returns the fully qualified name for the End of Record Event terminal. You can use this terminal as the source for a trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:End of Record:Terminal Name**
  - C Attribute: **NISCOPE\_ATTR\_END\_OF\_RECORD\_EVENT\_TERMINAL\_NAME**
-

## end\_of\_record\_to\_advance\_trigger\_holdoff

### `niscope.Session.end_of_record_to_advance_trigger_holdoff`

End of Record to Advance Trigger Holdoff is the length of time (in seconds) that a device waits between the completion of one record and the acquisition of pre-trigger samples for the next record. During this time, the acquisition engine state delays the transition to the Wait for Advance Trigger state, and will not store samples in onboard memory, accept an Advance Trigger, or trigger on the input signal.. **Supported Devices:** NI 5185/5186

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:End of Record to Advance Trigger Holdoff**
  - C Attribute: **NISCOPE\_ATTR\_END\_OF\_RECORD\_TO\_ADVANCE\_TRIGGER\_HOLDOFF**
- 

## equalization\_filter\_enabled

### `niscope.Session.equalization_filter_enabled`

Enables the onboard signal processing FIR block. This block is connected directly to the input signal. This filter is designed to compensate the input signal for artifacts introduced to the signal outside of the digitizer. However, since this is a generic FIR filter any coefficients are valid. Coefficients should be between +1 and -1 in value.

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].equalization_filter_enabled`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.equalization_filter_enabled`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Onboard Signal Processing:Equalization:Equalization Filter Enabled**

- C Attribute: `NISCOPE_ATTR_EQUALIZATION_FILTER_ENABLED`
- 

## `equalization_num_coefficients`

### `niscope.Session.equalization_num_coefficients`

Returns the number of coefficients that the FIR filter can accept. This filter is designed to compensate the input signal for artifacts introduced to the signal outside of the digitizer. However, since this is a generic FIR filter any coefficients are valid. Coefficients should be between +1 and -1 in value.

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].equalization_num_coefficients`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.equalization_num_coefficients`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Onboard Signal Processing:Equalization:Equalization Num Coefficients**
  - C Attribute: `NISCOPE_ATTR_EQUALIZATION_NUM_COEFFICIENTS`
- 

## `exported_advance_trigger_output_terminal`

### `niscope.Session.exported_advance_trigger_output_terminal`

Specifies the destination to export the advance trigger. When the advance trigger is received, the digitizer begins acquiring samples for the Nth record. Consult your device documentation for a specific list of valid destinations.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:



- LabVIEW Property: **Synchronization:Advance Trigger:Output Terminal**
- C Attribute: **NISCOPE\_ATTR\_EXPORTED\_ADVANCE\_TRIGGER\_OUTPUT\_TERMINAL**

### exported\_ref\_trigger\_output\_terminal

`niscope.Session.exported_ref_trigger_output_terminal`

Specifies the destination export for the reference (stop) trigger. Consult your device documentation for a specific list of valid destinations.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Output Terminal**
- C Attribute: **NISCOPE\_ATTR\_EXPORTED\_REF\_TRIGGER\_OUTPUT\_TERMINAL**

### exported\_start\_trigger\_output\_terminal

`niscope.Session.exported_start_trigger_output_terminal`

Specifies the destination to export the Start trigger. When the start trigger is received, the digitizer begins acquiring samples. Consult your device documentation for a specific list of valid destinations.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Start Trigger (Acq. Arm):Output Terminal**
- C Attribute: **NISCOPE\_ATTR\_EXPORTED\_START\_TRIGGER\_OUTPUT\_TERMINAL**

## flex\_fir\_antialias\_filter\_type

`niscope.Session.flex_fir_antialias_filter_type`

The NI 5922 flexible-resolution digitizer uses an onboard FIR lowpass antialias filter. Use this property to select from several types of filters to achieve desired filtering characteristics.

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].flex_fir_antialias_filter_type`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.flex_fir_antialias_filter_type`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.FlexFIRAntialiasFilterType</code>
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Advanced:Flex FIR Antialias Filter Type**
  - C Attribute: **NISCOPE\_ATTR\_FLEX\_FIR\_ANTIALIAS\_FILTER\_TYPE**
- 

## fpga\_bitfile\_path

`niscope.Session.fpga_bitfile_path`

Gets the absolute file path to the bitfile loaded on the FPGA.

---

**Note:** Gets the absolute file path to the bitfile loaded on the FPGA.

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>str</code>
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Device:FPGA Bitfile Path**
  - C Attribute: **NISCOPE\_ATTR\_FPGA\_BITFILE\_PATH**
-

## glitch\_condition

### `niscope.Session.glitch_condition`

Specifies whether the oscilloscope triggers on pulses of duration less than or greater than the value specified by the `niscope.Session.glitch_width` property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.GlitchCondition</code>
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_GLITCH\_CONDITION**

## glitch\_polarity

### `niscope.Session.glitch_polarity`

Specifies the polarity of pulses that trigger the oscilloscope for glitch triggering.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.GlitchPolarity</code>
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_GLITCH\_POLARITY**

## glitch\_width

### `niscope.Session.glitch_width`

Specifies the glitch duration, in seconds.

The oscilloscope triggers when it detects of pulse of duration either less than or greater than this value depending on the value of the `niscope.Session.glitch_condition` property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>float</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_GLITCH\_WIDTH**
- 

## high\_pass\_filter\_frequency

### `niscope.Session.high_pass_filter_frequency`

Specifies the frequency for the highpass filter in Hz. The device uses one of the valid values listed below. If an invalid value is specified, no coercion occurs. The default value is 0. **(PXIe-5164) Valid Values:** 0 90 450 **Related topics:** [Digital Filtering](#)

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].high_pass_filter_frequency`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.high_pass_filter_frequency`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Advanced:High Pass Filter Frequency**
  - C Attribute: **NISCOPE\_ATTR\_HIGH\_PASS\_FILTER\_FREQUENCY**
- 

## horz\_enforce\_realtime

### `niscope.Session.horz_enforce_realtime`

Indicates whether the digitizer enforces real-time measurements or allows equivalent-time measurements.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Enforce Realtime**
  - C Attribute: **NISCOPE\_ATTR\_HORZ\_ENFORCE\_REALTIME**
- 

## horz\_min\_num\_pts

### `niscope.Session.horz_min_num_pts`

Specifies the minimum number of points you require in the waveform record for each channel. NI-SCOPE uses the value you specify to configure the record length that the digitizer uses for waveform acquisition. `niscope.Session.horz_record_length` returns the actual record length. Valid Values: 1 - available onboard memory

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Min Number of Points**
  - C Attribute: **NISCOPE\_ATTR\_HORZ\_MIN\_NUM\_PTS**
- 

## horz\_num\_records

### `niscope.Session.horz_num_records`

Specifies the number of records to acquire. Can be used for multi-record acquisition and single-record acquisitions. Setting this to 1 indicates a single-record acquisition.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Number of Records**
  - C Attribute: **NISCOPE\_ATTR\_HORZ\_NUM\_RECORDS**
-

## horz\_record\_length

### `niscope.Session.horz_record_length`

Returns the actual number of points the digitizer acquires for each channel. The value is equal to or greater than the minimum number of points you specify with `niscope.Session.horz_min_num_pts`. Allocate a ViReal64 array of this size or greater to pass as the WaveformArray parameter of the Read and Fetch methods. This property is only valid after a call to the one of the Configure Horizontal methods.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Actual Record Length**
  - C Attribute: **NISCOPE\_ATTR\_HORZ\_RECORD\_LENGTH**
- 

## horz\_record\_ref\_position

### `niscope.Session.horz_record_ref_position`

Specifies the position of the Reference Event in the waveform record. When the digitizer detects a trigger, it waits the length of time the `niscope.Session.trigger_delay_time` property specifies. The event that occurs when the delay time elapses is the Reference Event. The Reference Event is relative to the start of the record and is a percentage of the record length. For example, the value 50.0 corresponds to the center of the waveform record and 0.0 corresponds to the first element in the waveform record. Valid Values: 0.0 - 100.0

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Reference Position**
  - C Attribute: **NISCOPE\_ATTR\_HORZ\_RECORD\_REF\_POSITION**
-

## horz\_sample\_rate

### `niscope.Session.horz_sample_rate`

Returns the effective sample rate using the current configuration. The units are samples per second. This property is only valid after a call to the one of the Configure Horizontal methods. Units: Hertz (Samples / Second)

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Actual Sample Rate**
  - C Attribute: **NISCOPE\_ATTR\_HORZ\_SAMPLE\_RATE**
- 

## horz\_time\_per\_record

### `niscope.Session.horz_time_per_record`

Specifies the length of time that corresponds to the record length. Units: Seconds

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	highitime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Advanced:Time Per Record**
  - C Attribute: **NISCOPE\_ATTR\_HORZ\_TIME\_PER\_RECORD**
- 

## input\_clock\_source

### `niscope.Session.input_clock_source`

Specifies the input source for the PLL reference clock (the 1 MHz to 20 MHz clock on the NI 5122, the 10 MHz clock for the NI 5112/5620/5621/5911) to which the digitizer will be phase-locked; for the NI 5102, this is the source of the board clock.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocking:Reference (Input) Clock Source**
  - C Attribute: **NISCOPE\_ATTR\_INPUT\_CLOCK\_SOURCE**
- 

## input\_impedance

`niscope.Session.input_impedance`

Specifies the input impedance for the channel in Ohms.

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].input_impedance`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.input_impedance`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Input Impedance**
  - C Attribute: **NISCOPE\_ATTR\_INPUT\_IMPEDANCE**
-



## instrument\_firmware\_revision

`niscope.Session.instrument_firmware_revision`

A string that contains the firmware revision information for the instrument you are currently using.

**Tip:** This property can be set/get on specific instruments within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container instruments to specify a subset.

Example: `my_session.instruments[ ... ].instrument_firmware_revision`

To set/get on all instruments, you can call the property directly on the `niscope.Session`.

Example: `my_session.instrument_firmware_revision`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	instruments

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Instrument Identification:Firmware Revision**
- C Attribute: **NISCOPE\_ATTR\_INSTRUMENT\_FIRMWARE\_REVISION**

## instrument\_manufacturer

`niscope.Session.instrument_manufacturer`

A string that contains the name of the instrument manufacturer.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Instrument Identification:Manufacturer**
- C Attribute: **NISCOPE\_ATTR\_INSTRUMENT\_MANUFACTURER**

## instrument\_model

`niscope.Session.instrument_model`

A string that contains the model number of the current instrument.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Instrument Identification:Model**
  - C Attribute: **NISCOPE\_ATTR\_INSTRUMENT\_MODEL**
- 

## interleaving\_offset\_correction\_enabled

`niscope.Session.interleaving_offset_correction_enabled`

Enables the interleaving offset correction on the specified channel. The default value is TRUE. **Related topics:** [Timed Interleaved Sampling](#)

---

**Note:** If disabled, warranted specifications are not guaranteed.

---

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].interleaving_offset_correction_enabled`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.interleaving_offset_correction_enabled`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Advanced:Interleaving Offset Correction Enabled**
  - C Attribute: **NISCOPE\_ATTR\_INTERLEAVING\_OFFSET\_CORRECTION\_ENABLED**
-

## io\_resource\_descriptor

### `niscope.Session.io_resource_descriptor`

Indicates the resource descriptor the driver uses to identify the physical device. If you initialize the driver with a logical name, this property contains the resource descriptor that corresponds to the entry in the IVI Configuration utility. If you initialize the instrument driver with the resource descriptor, this property contains that value. You can pass a logical name to `niscope.Session.Init()` or `niscope.Session.__init__()`. The IVI Configuration utility must contain an entry for the logical name. The logical name entry refers to a virtual instrument section in the IVI Configuration file. The virtual instrument section specifies a physical device and initial user options.

---

**Note:** One or more of the referenced methods are not in the Python API for this driver.

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Advanced Session Information:Resource Descriptor**
  - C Attribute: **NISCOPE\_ATTR\_IO\_RESOURCE\_DESCRIPTOR**
- 

## is\_probe\_comp\_on

### `niscope.Session.is_probe_comp_on`

---

**Tip:** This property can be set/get on specific instruments within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container instruments to specify a subset.

Example: `my_session.instruments[ ... ].is_probe_comp_on`

To set/get on all instruments, you can call the property directly on the `niscope.Session`.

Example: `my_session.is_probe_comp_on`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read only
Repeated Capabilities	instruments

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_IS\_PROBE\_COMP\_ON**
- 

## logical\_name

### `niscope.Session.logical_name`

A string containing the logical name you specified when opening the current IVI session. You can pass a logical name to `niscope.Session.Init()` or `niscope.Session.__init__()`. The IVI Configuration utility must contain an entry for the logical name. The logical name entry refers to a virtual instrument section in the IVI Configuration file. The virtual instrument section specifies a physical device and initial user options.

---

**Note:** One or more of the referenced methods are not in the Python API for this driver.

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Advanced Session Information:Logical Name**
  - C Attribute: **NISCOPE\_ATTR\_LOGICAL\_NAME**
- 

## master\_enable

### `niscope.Session.master_enable`

Specifies whether you want the device to be a master or a slave. The master typically originates the trigger signal and clock sync pulse. For a standalone device, set this property to False.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Master Enable**
- C Attribute: **NISCOPE\_ATTR\_MASTER\_ENABLE**

## max\_input\_frequency

### `niscope.Session.max_input_frequency`

Specifies the bandwidth of the channel. Express this value as the frequency at which the input circuitry attenuates the input signal by 3 dB. The units are hertz. Defined Values: `NISCOPE_VAL_BANDWIDTH_FULL` (-1.0) `NISCOPE_VAL_BANDWIDTH_DEVICE_DEFAULT` (0.0) `NISCOPE_VAL_20MHZ_BANDWIDTH` (20000000.0) `NISCOPE_VAL_100MHZ_BANDWIDTH` (100000000.0) `NISCOPE_VAL_20MHZ_MAX_INPUT_FREQUENCY` (20000000.0) `NISCOPE_VAL_100MHZ_MAX_INPUT_FREQUENCY` (100000000.0)

**Note:** One or more of the referenced values are not in the Python API for this driver. Enums that only define values, or represent True/False, have been removed.

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_input_frequency`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.max_input_frequency`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Maximum Input Frequency**
- C Attribute: **NISCOPE\_ATTR\_MAX\_INPUT\_FREQUENCY**

## max\_real\_time\_sampling\_rate

### `niscope.Session.max_real_time_sampling_rate`

Returns the maximum real time sample rate in Hz.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Maximum Real Time Sample Rate**
  - C Attribute: **NISCOPE\_ATTR\_MAX\_REAL\_TIME\_SAMPLING\_RATE**
- 

## max\_ris\_rate

`niscope.Session.max_ris_rate`

Returns the maximum sample rate in RIS mode in Hz.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Maximum RIS Rate**
  - C Attribute: **NISCOPE\_ATTR\_MAX\_RIS\_RATE**
- 

## meas\_array\_gain

`niscope.Session.meas_array_gain`

Every element of an array is multiplied by this scalar value during the Array Gain measurement. Refer to [ARRAY\\_GAIN](#) for more information. Default: 1.0

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_array_gain`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_array_gain`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Array Gain**
- C Attribute: **NISCOPE\_ATTR\_MEAS\_ARRAY\_GAIN**

## meas\_array\_offset

### `niscope.Session.meas_array_offset`

Every element of an array is added to this scalar value during the Array Offset measurement. Refer to [ARRAY\\_OFFSET](#) for more information. Default: 0.0

**Tip:** This property can be set/get on specific channels within your [niscope.Session](#) instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_array_offset`

To set/get on all channels, you can call the property directly on the [niscope.Session](#).

Example: `my_session.meas_array_offset`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Array Offset**
- C Attribute: **NISCOPE\_ATTR\_MEAS\_ARRAY\_OFFSET**

## meas\_chan\_high\_ref\_level

### `niscope.Session.meas_chan_high_ref_level`

Stores the high reference level used in many scalar measurements. Different channels may have different reference levels. Do not use the IVI-defined, nonchannel-based properties such as [niscope.Session.meas\\_high\\_ref](#) if you use this property to set various channels to different values. Default: 90%

**Tip:** This property can be set/get on specific channels within your [niscope.Session](#) instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_chan_high_ref_level`

To set/get on all channels, you can call the property directly on the [niscope.Session](#).

Example: `my_session.meas_chan_high_ref_level`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Reference Levels:Channel Based High Ref Level**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_CHAN\_HIGH\_REF\_LEVEL**
- 

## meas\_chan\_low\_ref\_level

`niscope.Session.meas_chan_low_ref_level`

Stores the low reference level used in many scalar measurements. Different channels may have different reference levels. Do not use the IVI-defined, nonchannel-based properties such as [`niscope.Session.meas\_low\_ref`](#) if you use this property to set various channels to different values. Default: 10%

---

**Tip:** This property can be set/get on specific channels within your [`niscope.Session`](#) instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_chan_low_ref_level`

To set/get on all channels, you can call the property directly on the [`niscope.Session`](#).

Example: `my_session.meas_chan_low_ref_level`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Reference Levels:Channel Based Low Ref Level**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_CHAN\_LOW\_REF\_LEVEL**
-



## meas\_chan\_mid\_ref\_level

### `niscope.Session.meas_chan_mid_ref_level`

Stores the mid reference level used in many scalar measurements. Different channels may have different reference levels. Do not use the IVI-defined, nonchannel-based properties such as `niscope.Session.meas_mid_ref` if you use this property to set various channels to different values. Default: 50%

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_chan_mid_ref_level`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_chan_mid_ref_level`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Reference Levels:Channel Based Mid Ref Level**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_CHAN\_MID\_REF\_LEVEL**
- 

## meas\_filter\_center\_freq

### `niscope.Session.meas_filter_center_freq`

The center frequency in hertz for filters of type bandpass and bandstop. The width of the filter is specified by `niscope.Session.meas_filter_width`, where the cutoff frequencies are the center  $\pm$  width. Default: 1.0e6 Hz

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_filter_center_freq`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_filter_center_freq`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:Center Frequency**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_FILTER\_CENTER\_FREQ**
- 

### `meas_filter_cutoff_freq`

`niscope.Session.meas_filter_cutoff_freq`

Specifies the cutoff frequency in hertz for filters of type lowpass and highpass. The cutoff frequency definition varies depending on the filter. Default: 1.0e6 Hz

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_filter_cutoff_freq`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_filter_cutoff_freq`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:Cutoff Frequency**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_FILTER\_CUTOFF\_FREQ**
-

## meas\_filter\_order

### `niscope.Session.meas_filter_order`

Specifies the order of an IIR filter. All positive integers are valid. Default: 2

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_filter_order`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_filter_order`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	channels

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:IIR Order**
- C Attribute: **NISCOPE\_ATTR\_MEAS\_FILTER\_ORDER**

## meas\_filter\_ripple

### `niscope.Session.meas_filter_ripple`

Specifies the amount of ripple in the passband in units of decibels (positive values). Used only for Chebyshev filters. The more ripple allowed gives a sharper cutoff for a given filter order. Default: 0.1 dB

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_filter_ripple`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_filter_ripple`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:Ripple**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_FILTER\_RIPPLE**
- 

## meas\_filter\_taps

### `niscope.Session.meas_filter_taps`

Defines the number of taps (coefficients) for an FIR filter. Default: 25

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_filter_taps`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_filter_taps`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:FIR Taps**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_FILTER\_TAPS**
- 

## meas\_filter\_transient\_waveform\_percent

### `niscope.Session.meas_filter_transient_waveform_percent`

The percentage (0 - 100%) of the IIR filtered waveform to eliminate from the beginning of the waveform. This allows eliminating the transient portion of the waveform that is undefined due to the assumptions necessary at the boundary condition. Default: 20.0%

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_filter_transient_waveform_percent`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_filter_transient_waveform_percent`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:Percent Waveform Transient**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_FILTER\_TRANSIENT\_WAVEFORM\_PERCENT**
- 

## meas\_filter\_type

### `niscope.Session.meas_filter_type`

Specifies the type of filter, for both IIR and FIR filters. The allowed values are the following:

- `NISCOPE_VAL_MEAS_LOWPASS`
- `NISCOPE_VAL_MEAS_HIGHPASS`
- `NISCOPE_VAL_MEAS_BANDPASS`
- `NISCOPE_VAL_MEAS_BANDSTOP`

Default: `NISCOPE_VAL_MEAS_LOWPASS`

---

**Note:** One or more of the referenced values are not in the Python API for this driver. Enums that only define values, or represent True/False, have been removed.

---



---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_filter_type`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_filter_type`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.FilterType</code>
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:Type**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_FILTER\_TYPE**
-

## meas\_filter\_width

### `niscope.Session.meas_filter_width`

Specifies the width of bandpass and bandstop type filters in hertz. The cutoff frequencies occur at `niscope.Session.meas_filter_center_freq`  $\pm$  one-half width. Default: 1.0e3 Hz

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_filter_width`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_filter_width`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:Width**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_FILTER\_WIDTH**
- 

## meas\_fir\_filter\_window

### `niscope.Session.meas_fir_filter_window`

Specifies the FIR window type. The possible choices are: `NONE` `HANNING_WINDOW` `HAMMING_WINDOW` `TRIANGLE_WINDOW` `FLAT_TOP_WINDOW` `BLACKMAN_WINDOW` The symmetric windows are applied to the FIR filter coefficients to limit passband ripple in FIR filters. Default: `NONE`

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_fir_filter_window`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_fir_filter_window`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.FIRFilterWindow
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Filter:FIR Window**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_FIR\_FILTER\_WINDOW**
- 

## meas\_high\_ref

`niscope.Session.meas_high_ref`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_MEAS\_HIGH\_REF**
- 

## meas\_hysteresis\_percent

`niscope.Session.meas_hysteresis_percent`

Digital hysteresis that is used in several of the scalar waveform measurements. This property specifies the percentage of the full-scale vertical range for the hysteresis window size. Default: 2%

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_hysteresis_percent`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_hysteresis_percent`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Hysteresis Percent**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_HYSTERESIS\_PERCENT**
- 

## meas\_interpolation\_sampling\_factor

### `niscope.Session.meas_interpolation_sampling_factor`

The new number of points for polynomial interpolation is the sampling factor times the input number of points. For example, if you acquire 1,000 points with the digitizer and set this property to 2.5, calling `niscope.Session.FetchWaveformMeasurementArray()` with the *POLYNOMIAL\_INTERPOLATION* measurement resamples the waveform to 2,500 points. Default: 2.0

---

**Note:** One or more of the referenced methods are not in the Python API for this driver.

---

---

**Tip:** This property can be set/get on specific channels within your *niscope.Session* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_interpolation_sampling_factor`

To set/get on all channels, you can call the property directly on the *niscope.Session*.

Example: `my_session.meas_interpolation_sampling_factor`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Interpolation:Sampling Factor**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_INTERPOLATION\_SAMPLING\_FACTOR**
- 

## meas\_last\_acq\_histogram\_size

### `niscope.Session.meas_last_acq_histogram_size`

Specifies the size (that is, the number of bins) in the last acquisition histogram. This histogram is used to determine several scalar measurements, most importantly voltage low and voltage high. Default: 256

---

**Tip:** This property can be set/get on specific channels within your *niscope.Session* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.



Example: `my_session.channels[ ... ].meas_last_acq_histogram_size`

To set/get on all channels, you can call the property directly on the [`niscope.Session`](#).

Example: `my_session.meas_last_acq_histogram_size`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	channels

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Last Acq. Histogram Size**
- C Attribute: **NISCOPE\_ATTR\_MEAS\_LAST\_ACQ\_HISTOGRAM\_SIZE**

## meas\_low\_ref

`niscope.Session.meas_low_ref`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_MEAS\_LOW\_REF**

## meas\_mid\_ref

`niscope.Session.meas_mid_ref`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: `NISCOPE_ATTR_MEAS_MID_REF`
- 

## meas\_other\_channel

### `niscope.Session.meas_other_channel`

Specifies the second channel for two-channel measurements, such as [ADD\\_CHANNELS](#). If processing steps are registered with this channel, the processing is done before the waveform is used in a two-channel measurement. Default: '0'

---

**Tip:** This property can be set/get on specific channels within your [niscope.Session](#) instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_other_channel`

To set/get on all channels, you can call the property directly on the [niscope.Session](#).

Example: `my_session.meas_other_channel`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str or int
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Other Channel**
  - C Attribute: `NISCOPE_ATTR_MEAS_OTHER_CHANNEL`
- 

## meas\_percentage\_method

### `niscope.Session.meas_percentage_method`

Specifies the method used to map percentage reference units to voltages for the reference. Possible values are: `NISCOPE_VAL_MEAS_LOW_HIGH` `NISCOPE_VAL_MEAS_MIN_MAX` `NISCOPE_VAL_MEAS_BASE_TOP` Default: `NISCOPE_VAL_MEAS_BASE_TOP`

---

**Note:** One or more of the referenced values are not in the Python API for this driver. Enums that only define values, or represent True/False, have been removed.

---



---

**Tip:** This property can be set/get on specific channels within your [niscope.Session](#) instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_percentage_method`

To set/get on all channels, you can call the property directly on the [niscope.Session](#).

---

Example: `my_session.meas_percentage_method`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.PercentageMethod</code>
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Reference Levels:Percentage Units Method**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_PERCENTAGE\_METHOD**
- 

## meas\_polynomial\_interpolation\_order

`niscope.Session.meas_polynomial_interpolation_order`

Specifies the polynomial order used for the polynomial interpolation measurement. For example, an order of 1 is linear interpolation whereas an order of 2 specifies parabolic interpolation. Any positive integer is valid. Default: 1

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_polynomial_interpolation_order`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_polynomial_interpolation_order`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>int</code>
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Interpolation:Polynomial Interpolation Order**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_POLYNOMIAL\_INTERPOLATION\_ORDER**
-

## meas\_ref\_level\_units

### `niscope.Session.meas_ref_level_units`

Specifies the units of the reference levels. `NISCOPE_VAL_MEAS_VOLTAGE`—Specifies that the reference levels are given in units of volts `NISCOPE_VAL_MEAS_PERCENTAGE`—Percentage units, where the measurements voltage low and voltage high represent 0% and 100%, respectively. Default: `NISCOPE_VAL_MEAS_PERCENTAGE`

---

**Note:** One or more of the referenced values are not in the Python API for this driver. Enums that only define values, or represent True/False, have been removed.

---

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_ref_level_units`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_ref_level_units`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.RefLevelUnits</code>
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Reference Levels:Units**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_REF\_LEVEL\_UNITS**
- 

## meas\_time\_histogram\_high\_time

### `niscope.Session.meas_time_histogram_high_time`

Specifies the highest time value included in the multiple acquisition time histogram. The units are always seconds. Default: 5.0e-4 seconds

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_time_histogram_high_time`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_time_histogram_high_time`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Time Histogram:High Time**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_TIME\_HISTOGRAM\_HIGH\_TIME**
- 

## meas\_time\_histogram\_high\_volts

`niscope.Session.meas_time_histogram_high_volts`

Specifies the highest voltage value included in the multiple-acquisition time histogram. The units are always volts. Default: 10.0 V

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_time_histogram_high_volts`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_time_histogram_high_volts`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Time Histogram:High Volts**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_TIME\_HISTOGRAM\_HIGH\_VOLTS**
-

## meas\_time\_histogram\_low\_time

`niscope.Session.meas_time_histogram_low_time`

Specifies the lowest time value included in the multiple-acquisition time histogram. The units are always seconds. Default:  $-5.0\text{e-}4$  seconds

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_time_histogram_low_time`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_time_histogram_low_time`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Time Histogram:Low Time**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_TIME\_HISTOGRAM\_LOW\_TIME**
- 

## meas\_time\_histogram\_low\_volts

`niscope.Session.meas_time_histogram_low_volts`

Specifies the lowest voltage value included in the multiple acquisition time histogram. The units are always volts. Default:  $-10.0\text{ V}$

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_time_histogram_low_volts`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_time_histogram_low_volts`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Time Histogram:Low Volts**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_TIME\_HISTOGRAM\_LOW\_VOLTS**
- 

## meas\_time\_histogram\_size

`niscope.Session.meas_time_histogram_size`

Determines the multiple acquisition voltage histogram size. The size is set during the first call to a time histogram measurement after clearing the measurement history with `niscope.Session.clear_waveform_measurement_stats()`. Default: 256

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_time_histogram_size`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_time_histogram_size`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Time Histogram:Size**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_TIME\_HISTOGRAM\_SIZE**
- 

## meas\_voltage\_histogram\_high\_volts

`niscope.Session.meas_voltage_histogram_high_volts`

Specifies the highest voltage value included in the multiple acquisition voltage histogram. The units are always volts. Default: 10.0 V

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_voltage_histogram_high_volts`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_voltage_histogram_high_volts`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Voltage Histogram:High Volts**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_VOLTAGE\_HISTOGRAM\_HIGH\_VOLTS**
- 

### `meas_voltage_histogram_low_volts`

`niscope.Session.meas_voltage_histogram_low_volts`

Specifies the lowest voltage value included in the multiple-acquisition voltage histogram. The units are always volts. Default: -10.0 V

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_voltage_histogram_low_volts`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_voltage_histogram_low_volts`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Voltage Histogram:Low Volts**
  - C Attribute: **NISCOPE\_ATTR\_MEAS\_VOLTAGE\_HISTOGRAM\_LOW\_VOLTS**
-



## meas\_voltage\_histogram\_size

### `niscope.Session.meas_voltage_histogram_size`

Determines the multiple acquisition voltage histogram size. The size is set the first time a voltage histogram measurement is called after clearing the measurement history with the method `niscope.Session.clear_waveform_measurement_stats()`. Default: 256

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].meas_voltage_histogram_size`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.meas_voltage_histogram_size`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	channels

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Measurement:Voltage Histogram:Size**
- C Attribute: **NISCOPE\_ATTR\_MEAS\_VOLTAGE\_HISTOGRAM\_SIZE**

## min\_sample\_rate

### `niscope.Session.min_sample_rate`

Specify the sampling rate for the acquisition in Samples per second. Valid Values: The combination of sampling rate and min record length must allow the digitizer to sample at a valid sampling rate for the acquisition type specified in `niscope.Session.ConfigureAcquisition()` and not require more memory than the onboard memory module allows.

**Note:** One or more of the referenced methods are not in the Python API for this driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Min Sample Rate**
  - C Attribute: **NISCOPE\_ATTR\_MIN\_SAMPLE\_RATE**
- 

## onboard\_memory\_size

### `niscope.Session.onboard_memory_size`

Returns the total combined amount of onboard memory for all channels in bytes.

---

**Tip:** This property can be set/get on specific instruments within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container instruments to specify a subset.

Example: `my_session.instruments[ ... ].onboard_memory_size`

To set/get on all instruments, you can call the property directly on the `niscope.Session`.

Example: `my_session.onboard_memory_size`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	instruments

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Memory Size**
  - C Attribute: **NISCOPE\_ATTR\_ONBOARD\_MEMORY\_SIZE**
- 

## output\_clock\_source

### `niscope.Session.output_clock_source`

Specifies the output source for the 10 MHz clock to which another digitizer's sample clock can be phased-locked.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocking:Output Clock Source**

- C Attribute: **NISCOPE\_ATTR\_OUTPUT\_CLOCK\_SOURCE**

## pll\_lock\_status

### `niscope.Session.pll_lock_status`

If TRUE, the PLL has remained locked to the external reference clock since it was last checked. If FALSE, the PLL has become unlocked from the external reference clock since it was last checked.

**Tip:** This property can be set/get on specific instruments within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container instruments to specify a subset.

Example: `my_session.instruments[ ... ].pll_lock_status`

To set/get on all instruments, you can call the property directly on the `niscope.Session`.

Example: `my_session.pll_lock_status`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read only
Repeated Capabilities	instruments

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocking:PLL Lock Status**
- C Attribute: **NISCOPE\_ATTR\_PLL\_LOCK\_STATUS**

## points\_done

### `niscope.Session.points_done`

Actual number of samples acquired in the record specified by `niscope.Session.fetch_record_number` from the `niscope.Session.fetch_relative_to` and `niscope.Session.fetch_offset` properties.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Fetch:Points Done**

- C Attribute: **NISCOPE\_ATTR\_POINTS\_DONE**
- 

## **poll\_interval**

### **`niscope.Session.poll_interval`**

Specifies the poll interval in milliseconds to use during RIS acquisitions to check whether the acquisition is complete.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_POLL\_INTERVAL**
- 

## **probe\_attenuation**

### **`niscope.Session.probe_attenuation`**

Specifies the probe attenuation for the input channel. For example, for a 10:1 probe, set this property to 10.0. Valid Values: Any positive real number. Typical values are 1, 10, and 100.

---

**Tip:** This property can be set/get on specific channels within your *`niscope.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].probe_attenuation`

To set/get on all channels, you can call the property directly on the *`niscope.Session`*.

Example: `my_session.probe_attenuation`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Probe Attenuation**
  - C Attribute: **NISCOPE\_ATTR\_PROBE\_ATTENUATION**
-

## ready\_for\_advance\_event\_output\_terminal

### `niscope.Session.ready_for_advance_event_output_terminal`

Specifies the destination for the Ready for Advance Event. When this event is asserted, the digitizer is ready to receive an advance trigger. Consult your device documentation for a specific list of valid destinations.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Ready for Advance:Output Terminal**
  - C Attribute: **NISCOPE\_ATTR\_READY\_FOR\_ADVANCE\_EVENT\_OUTPUT\_TERMINAL**
- 

## ready\_for\_advance\_event\_terminal\_name

### `niscope.Session.ready_for_advance_event_terminal_name`

Returns the fully qualified name for the Ready for Advance Event terminal. You can use this terminal as the source for a trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Ready for Advance:Terminal Name**
  - C Attribute: **NISCOPE\_ATTR\_READY\_FOR\_ADVANCE\_EVENT\_TERMINAL\_NAME**
-

## ready\_for\_ref\_event\_output\_terminal

### `niscope.Session.ready_for_ref_event_output_terminal`

Specifies the destination for the Ready for Reference Event. When this event is asserted, the digitizer is ready to receive a reference trigger. Consult your device documentation for a specific list of valid destinations.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Ready for Reference:Output Terminal**
  - C Attribute: **NISCOPE\_ATTR\_READY\_FOR\_REF\_EVENT\_OUTPUT\_TERMINAL**
- 

## ready\_for\_ref\_event\_terminal\_name

### `niscope.Session.ready_for_ref_event_terminal_name`

Returns the fully qualified name for the Ready for Reference Event terminal. You can use this terminal as the source for a trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Ready for Reference:Terminal Name**
  - C Attribute: **NISCOPE\_ATTR\_READY\_FOR\_REF\_EVENT\_TERMINAL\_NAME**
-

## ready\_for\_start\_event\_output\_terminal

### `niscope.Session.ready_for_start_event_output_terminal`

Specifies the destination for the Ready for Start Event. When this event is asserted, the digitizer is ready to receive a start trigger. Consult your device documentation for a specific list of valid destinations.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Ready for Start:Output Terminal**
  - C Attribute: **NISCOPE\_ATTR\_READY\_FOR\_START\_EVENT\_OUTPUT\_TERMINAL**
- 

## ready\_for\_start\_event\_terminal\_name

### `niscope.Session.ready_for_start_event_terminal_name`

Returns the fully qualified name for the Ready for Start Event terminal. You can use this terminal as the source for a trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Ready for Start:Terminal Name**
  - C Attribute: **NISCOPE\_ATTR\_READY\_FOR\_START\_EVENT\_TERMINAL\_NAME**
-

## records\_done

`niscope.Session.records_done`

Specifies the number of records that have been completely acquired.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Fetch:Records Done**
  - C Attribute: **NISCOPE\_ATTR\_RECORDS\_DONE**
- 

## record\_arm\_source

`niscope.Session.record_arm_source`

Specifies the record arm source.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Record Arm Source**
  - C Attribute: **NISCOPE\_ATTR\_RECORD\_ARM\_SOURCE**
- 

## ref\_clk\_rate

`niscope.Session.ref_clk_rate`

If `niscope.Session.input_clock_source` is an external source, this property specifies the frequency of the input, or reference clock, to which the internal sample clock timebase is synchronized. The frequency is in hertz.

The following table lists the characteristics of this property.



Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocking:Reference Clock Rate**
  - C Attribute: **NISCOPE\_ATTR\_REF\_CLK\_RATE**
- 

## ref\_trigger\_detector\_location

`niscope.Session.ref_trigger_detector_location`

Indicates which analog compare circuitry to use on the device.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.RefTriggerDetectorLocation</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Onboard Signal Processing:Ref Trigger Detection Location**
  - C Attribute: **NISCOPE\_ATTR\_REF\_TRIGGER\_DETECTOR\_LOCATION**
- 

## ref\_trigger\_minimum\_quiet\_time

`niscope.Session.ref_trigger_minimum_quiet_time`

The amount of time the trigger circuit must not detect a signal above the trigger level before the trigger is armed. This property is useful for triggering at the beginning and not in the middle of signal bursts.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>hightime.timedelta</code> , <code>datetime.timedelta</code> , or float in seconds
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Onboard Signal Processing:Ref Trigger Min Quiet Time**
  - C Attribute: **NISCOPE\_ATTR\_REF\_TRIGGER\_MINIMUM\_QUIET\_TIME**
- 

## ref\_trigger\_terminal\_name

`niscope.Session.ref_trigger_terminal_name`

Returns the fully qualified name for the Reference Trigger terminal. You can use this terminal as the source for another trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Terminal Name**
  - C Attribute: **NISCOPE\_ATTR\_REF\_TRIGGER\_TERMINAL\_NAME**
- 

## ref\_trig\_tdc\_enable

`niscope.Session.ref_trig_tdc_enable`

This property controls whether the TDC is used to compute an accurate trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:Advanced:Enable TDC**
  - C Attribute: **NISCOPE\_ATTR\_REF\_TRIG\_TDC\_ENABLE**
-

## resolution

### `niscope.Session.resolution`

Indicates the bit width of valid data (as opposed to padding bits) in the acquired waveform. Compare to `niscope.Session.binary_sample_width`.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Acquisition:Resolution**
  - C Attribute: **NISCOPE\_ATTR\_RESOLUTION**
- 

## ris\_in\_auto\_setup\_enable

### `niscope.Session.ris_in_auto_setup_enable`

Indicates whether the digitizer should use RIS sample rates when searching for a frequency in autotsetup. Valid Values: True (1) - Use RIS sample rates in autotsetup False (0) - Do not use RIS sample rates in autotsetup

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Acquisition:Advanced:Enable RIS in Auto Setup**
  - C Attribute: **NISCOPE\_ATTR\_RIS\_IN\_AUTO\_SETUP\_ENABLE**
-

## ris\_method

### `niscope.Session.ris_method`

Specifies the algorithm for random-interleaved sampling, which is used if the sample rate exceeds the value of `niscope.Session.max_real_time_sampling_rate`.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.RISMethod</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:RIS Method**
  - C Attribute: **NISCOPE\_ATTR\_RIS\_METHOD**
- 

## ris\_num\_averages

### `niscope.Session.ris_num_averages`

The number of averages for each bin in an RIS acquisition. The number of averages times the over-sampling factor is the minimum number of real-time acquisitions necessary to reconstruct the RIS waveform. Averaging is useful in RIS because the trigger times are not evenly spaced, so adjacent points in the reconstructed waveform not be accurately spaced. By averaging, the errors in both time and voltage are smoothed.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>int</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Horizontal:RIS Num Avg**
  - C Attribute: **NISCOPE\_ATTR\_RIS\_NUM\_AVERAGES**
-

## run\_high\_threshold

### `niscope.Session.run_high_threshold`

Specifies the higher of two thresholds, in volts, that bound the vertical range to examine for runt pulses.

The runt threshold that causes the oscilloscope to trigger depends on the runt polarity you select. Refer to the [`niscope.Session.run\_polarity`](#) property for more information.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_RUNT\_HIGH\_THRESHOLD**
- 

## run\_low\_threshold

### `niscope.Session.run_low_threshold`

Specifies the lower of two thresholds, in volts, that bound the vertical range to examine for runt pulses.

The runt threshold that causes the oscilloscope to trigger depends on the runt polarity you select. Refer to the [`niscope.Session.run\_polarity`](#) property for more information.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_RUNT\_LOW\_THRESHOLD**
-

## run\_t\_polarity

### `niscope.Session.run_t_polarity`

Specifies the polarity of pulses that trigger the oscilloscope for runt triggering.

When set to **POSITIVE**, the oscilloscope triggers when the following conditions are met:

- The leading edge of a pulse crosses the `niscope.Session.run_t_low_threshold` in a positive direction;
- The trailing edge of the pulse crosses the `niscope.Session.run_t_low_threshold` in a negative direction; and
- No portion of the pulse crosses the `niscope.Session.run_t_high_threshold`.

When set to **NEGATIVE**, the oscilloscope triggers when the following conditions are met:

- The leading edge of a pulse crosses the `niscope.Session.run_t_high_threshold` in a negative direction;
- The trailing edge of the pulse crosses the `niscope.Session.run_t_high_threshold` in a positive direction; and
- No portion of the pulse crosses the `niscope.Session.run_t_low_threshold`.

When set to **EITHER**, the oscilloscope triggers in either case.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.Run_tPolarity</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_RUN\_T\_POLARITY**
- 

## run\_t\_time\_condition

### `niscope.Session.run_t_time_condition`

Specifies whether runt triggers are time qualified, and if so, how the oscilloscope triggers in relation to the duration range bounded by the `niscope.Session.run_t_time_low_limit` and `niscope.Session.run_t_time_high_limit` properties.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.Run_tTimeCondition</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_RUNT\_TIME\_CONDITION**
- 

## **runt\_time\_high\_limit**

`niscope.Session.runt_time_high_limit`

Specifies, in seconds, the high runt threshold time.

This property sets the upper bound on the duration of runt pulses that may trigger the oscilloscope. The `niscope.Session.runt_time_condition` property determines how the oscilloscope triggers in relation to the runt time limits.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_RUNT\_TIME\_HIGH\_LIMIT**
- 

## **runt\_time\_low\_limit**

`niscope.Session.runt_time_low_limit`

Specifies, in seconds, the low runt threshold time.

This property sets the lower bound on the duration of runt pulses that may trigger the oscilloscope. The `niscope.Session.runt_time_condition` property determines how the oscilloscope triggers in relation to the runt time limits.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_RUNT\_TIME\_LOW\_LIMIT**
-

## sample\_mode

`niscope.Session.sample_mode`

Indicates the sample mode the digitizer is currently using.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Acquisition:Sample Mode**
  - C Attribute: **NISCOPE\_ATTR\_SAMPLE\_MODE**
- 

## samp\_clk\_timebase\_div

`niscope.Session.samp_clk_timebase_div`

If `niscope.Session.samp_clk_timebase_src` is an external source, specifies the ratio between the sample clock timebase rate and the actual sample rate, which can be slower.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocking:Sample Clock Timebase Divisor**
  - C Attribute: **NISCOPE\_ATTR\_SAMP\_CLK\_TIMEBASE\_DIV**
- 

## sample\_clock\_timebase\_multiplier

`niscope.Session.sample_clock_timebase_multiplier`

If `niscope.Session.samp_clk_timebase_src` is an external source, this property specifies the ratio between the `niscope.Session.samp_clk_timebase_rate` and the actual sample rate, which can be higher. This property can be used in conjunction with `niscope.Session.samp_clk_timebase_div`. Some devices use multiple ADCs to sample the same channel at an effective sample rate that is greater than the specified clock rate. When providing an external sample clock use this property to indicate when you want a higher sample rate. Valid values for this property vary by device and current configuration.



**Related topics:** [Sample Clock](#)

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_SAMP\_CLK\_TIMEBASE\_MULT**
- 

### samp\_clk\_timebase\_rate

`niscope.Session.samp_clk_timebase_rate`

If `niscope.Session.samp_clk_timebase_src` is an external source, specifies the frequency in hertz of the external clock used as the timebase source.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocking:Sample Clock Timebase Rate**
  - C Attribute: **NISCOPE\_ATTR\_SAMP\_CLK\_TIMEBASE\_RATE**
- 

### samp\_clk\_timebase\_src

`niscope.Session.samp_clk_timebase_src`

Specifies the source of the sample clock timebase, which is the timebase used to control waveform sampling. The actual sample rate may be the timebase itself or a divided version of the timebase, depending on the `niscope.Session.min_sample_rate` (for internal sources) or the `niscope.Session.samp_clk_timebase_div` (for external sources).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocking:Sample Clock Timebase Source**
  - C Attribute: **NISCOPE\_ATTR\_SAMP\_CLK\_TIMEBASE\_SRC**
- 

## serial\_number

`niscope.Session.serial_number`

Returns the serial number of the device.

---

**Tip:** This property can be set/get on specific instruments within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container instruments to specify a subset.

Example: `my_session.instruments[ ... ].serial_number`

To set/get on all instruments, you can call the property directly on the `niscope.Session`.

Example: `my_session.serial_number`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	instruments

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Device:Serial Number**
  - C Attribute: **NISCOPE\_ATTR\_SERIAL\_NUMBER**
- 

## accessory\_gain

`niscope.Session.accessory_gain`

Returns the calibration gain for the current device configuration.

**Related topics:** [NI 5122/5124/5142 Calibration](#)

---

**Note:** This property is supported only by the NI PXI-5900 differential amplifier.

---



---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].accessory_gain`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

---

Example: `my_session.accessory_gain`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_SIGNAL\_COND\_GAIN**
- 

## accessory\_offset

`niscope.Session.accessory_offset`

Returns the calibration offset for the current device configuration.

**Related topics:** [NI 5122/5124/5142 Calibration](#)

---

**Note:** This property is supported only by the NI PXI-5900 differential amplifier.

---



---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].accessory_offset`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.accessory_offset`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_SIGNAL\_COND\_OFFSET**
-

## simulate

### `niscope.Session.simulate`

Specifies whether or not to simulate instrument driver I/O operations. If simulation is enabled, instrument driver methods perform range checking and call `Ivi_GetAttribute` and `Ivi_SetAttribute` methods, but they do not perform instrument I/O. For output parameters that represent instrument data, the instrument driver methods return calculated values. The default value is `False`. Use the `niscope.Session.__init__()` method to override this value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes>User Options:Simulate**
  - C Attribute: **NISCOPE\_ATTR\_SIMULATE**
- 

## specific\_driver\_description

### `niscope.Session.specific_driver_description`

A string that contains a brief description of the specific driver

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Driver Identification:Description**
  - C Attribute: **NISCOPE\_ATTR\_SPECIFIC\_DRIVER\_DESCRIPTION**
-

## specific\_driver\_revision

`niscope.Session.specific_driver_revision`

A string that contains additional version information about this instrument driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Driver Identification:Revision**
  - C Attribute: **NISCOPE\_ATTR\_SPECIFIC\_DRIVER\_REVISION**
- 

## specific\_driver\_vendor

`niscope.Session.specific_driver_vendor`

A string that contains the name of the vendor that supplies this driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Driver Identification:Driver Vendor**
  - C Attribute: **NISCOPE\_ATTR\_SPECIFIC\_DRIVER\_VENDOR**
- 

## start\_to\_ref\_trigger\_holdoff

`niscope.Session.start_to_ref_trigger_holdoff`

Pass the length of time you want the digitizer to wait after it starts acquiring data until the digitizer enables the trigger system to detect a reference (stop) trigger. Units: Seconds Valid Values: 0.0 - 171.8

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Start To Ref Trigger Holdoff**
  - C Attribute: **NISCOPE\_ATTR\_START\_TO\_REF\_TRIGGER\_HOLDOFF**
- 

## start\_trigger\_terminal\_name

`niscope.Session.start_trigger_terminal_name`

Returns the fully qualified name for the Start Trigger terminal. You can use this terminal as the source for another trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Synchronization:Start Trigger (Acq. Arm):Terminal Name**
  - C Attribute: **NISCOPE\_ATTR\_START\_TRIGGER\_TERMINAL\_NAME**
- 

## supported\_instrument\_models

`niscope.Session.supported_instrument_models`

A string that contains a comma-separated list of the instrument model numbers supported by this driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Inherent IVI Attributes:Driver Capabilities:Supported Instrument Models**
  - C Attribute: **NISCOPE\_ATTR\_SUPPORTED\_INSTRUMENT\_MODELS**
- 

### trigger\_auto\_triggered

#### `niscope.Session.trigger_auto_triggered`

Specifies if the last acquisition was auto triggered. You can use the Auto Triggered property to find out if the last acquisition was triggered.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read only
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Auto Triggered**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_AUTO\_TRIGGERED**
- 

### trigger\_coupling

#### `niscope.Session.trigger_coupling`

Specifies how the digitizer couples the trigger source. This property affects instrument operation only when `niscope.Session.trigger_type` is set to *EDGE*, *HYSTERESIS*, or *WINDOW*.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.TriggerCoupling</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Coupling**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_COUPLING**
-

## trigger\_delay\_time

### `niscope.Session.trigger_delay_time`

Specifies the trigger delay time in seconds. The trigger delay time is the length of time the digitizer waits after it receives the trigger. The event that occurs when the trigger delay elapses is the Reference Event. Valid Values: 0.0 - 171.8

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Delay**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_DELAY\_TIME**
- 

## trigger\_holdoff

### `niscope.Session.trigger_holdoff`

Specifies the length of time (in seconds) the digitizer waits after detecting a trigger before enabling the trigger subsystem to detect another trigger. This property affects instrument operation only when the digitizer requires multiple acquisitions to build a complete waveform. The digitizer requires multiple waveform acquisitions when it uses equivalent-time sampling or when the digitizer is configured for a multi-record acquisition through a call to [\*`niscope.Session.configure\_horizontal\_timing\(\)`\*](#). Valid Values: 0.0 - 171.8

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Holdoff**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_HOLDOFF**
-



## trigger\_hysteresis

### `niscope.Session.trigger_hysteresis`

Specifies the size of the hysteresis window on either side of the trigger level. The digitizer triggers when the trigger signal passes through the threshold you specify with the Trigger Level parameter, has the slope you specify with the Trigger Slope parameter, and passes through the hysteresis window that you specify with this parameter.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Hysteresis**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_HYSTERESIS**
- 

## trigger\_impedance

### `niscope.Session.trigger_impedance`

Specifies the input impedance for the external analog trigger channel in Ohms. Valid Values: 50 - 50 ohms 1000000 - 1 mega ohm

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Impedance**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_IMPEDANCE**
-

## trigger\_level

### `niscope.Session.trigger_level`

Specifies the voltage threshold for the trigger subsystem. The units are volts. This property affects instrument behavior only when the `niscope.Session.trigger_type` is set to `EDGE`, `HYSTERESIS`, or `WINDOW`. Valid Values: The values of the range and offset parameters in `niscope.Session.configure_vertical()` determine the valid range for the trigger level on the channel you use as the Trigger Source. The value you pass for this parameter must meet the following conditions:

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Level**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_LEVEL**
- 

## trigger\_modifier

### `niscope.Session.trigger_modifier`

Configures the device to automatically complete an acquisition if a trigger has not been received. Valid Values: None (1) - Normal triggering Auto Trigger (2) - Auto trigger acquisition if no trigger arrives

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.TriggerModifier</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Modifier**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_MODIFIER**
-

## trigger\_slope

### `niscope.Session.trigger_slope`

Specifies if a rising or a falling edge triggers the digitizer. This property affects instrument operation only when `niscope.Session.trigger_type` is set to `EDGE`, `HYSTERESIS`, or `WINDOW`.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.TriggerSlope</code>
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Slope**
- C Attribute: **NISCOPE\_ATTR\_TRIGGER\_SLOPE**

## trigger\_source

### `niscope.Session.trigger_source`

Specifies the source the digitizer monitors for the trigger event.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>str</code>
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Source**
- C Attribute: **NISCOPE\_ATTR\_TRIGGER\_SOURCE**

## trigger\_type

### `niscope.Session.trigger_type`

Specifies the type of trigger to use.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TriggerType
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Type**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_TYPE**
- 

### trigger\_window\_high\_level

#### `niscope.Session.trigger_window_high_level`

Pass the upper voltage threshold you want the digitizer to use for window triggering. The digitizer triggers when the trigger signal enters or leaves the window you specify with `niscope.Session.trigger_window_low_level` and `niscope.Session.trigger_window_high_level`. Valid Values: The values of the Vertical Range and Vertical Offset parameters in `niscope.Session.configure_vertical()` determine the valid range for the High Window Level on the channel you use as the Trigger Source parameter in `niscope.Session.ConfigureTriggerSource()`. The value you pass for this parameter must meet the following conditions. High Trigger Level  $\leq$  Vertical Range/2 + Vertical Offset High Trigger Level  $\geq$  (-Vertical Range/2) + Vertical Offset High Trigger Level > Low Trigger Level

---

**Note:** One or more of the referenced methods are not in the Python API for this driver.

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Window:High Level**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_WINDOW\_HIGH\_LEVEL**
-

## trigger\_window\_low\_level

### `niscope.Session.trigger_window_low_level`

Pass the lower voltage threshold you want the digitizer to use for window triggering. The digitizer triggers when the trigger signal enters or leaves the window you specify with `niscope.Session.trigger_window_low_level` and `niscope.Session.trigger_window_high_level`. Units: Volts Valid Values: The values of the Vertical Range and Vertical Offset parameters in `niscope.Session.configure_vertical()` determine the valid range for the Low Window Level on the channel you use as the Trigger Source parameter in `niscope.Session.ConfigureTriggerSource()`. The value you pass for this parameter must meet the following conditions. Low Trigger Level  $\leq$  Vertical Range/2 + Vertical Offset Low Trigger Level  $\geq$  (-Vertical Range/2) + Vertical Offset Low Trigger Level < High Trigger Level

---

**Note:** One or more of the referenced methods are not in the Python API for this driver.

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Window:Low Level**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_WINDOW\_LOW\_LEVEL**
- 

## trigger\_window\_mode

### `niscope.Session.trigger_window_mode`

Specifies whether you want a trigger to occur when the signal enters or leaves the window specified by `niscope.Session.trigger_window_low_level`, or `niscope.Session.trigger_window_high_level`.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.TriggerWindowMode</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Window:Window Mode**
  - C Attribute: **NISCOPE\_ATTR\_TRIGGER\_WINDOW\_MODE**
-

## tv\_trigger\_event

`niscope.Session.tv_trigger_event`

Specifies the condition in the video signal that causes the digitizer to trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.VideoTriggerEvent</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Video:Event**
  - C Attribute: **NISCOPE\_ATTR\_TV\_TRIGGER\_EVENT**
- 

## tv\_trigger\_line\_number

`niscope.Session.tv_trigger_line_number`

Specifies the line on which to trigger, if [\*niscope.Session.tv\\_trigger\\_event\*](#) is set to line number. The valid ranges of the property depend on the signal format selected. M-NTSC has a valid range of 1 to 525. B/G-PAL, SECAM, 576i, and 576p have a valid range of 1 to 625. 720p has a valid range of 1 to 750. 1080i and 1080p have a valid range of 1125.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>int</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Video:Line Number**
  - C Attribute: **NISCOPE\_ATTR\_TV\_TRIGGER\_LINE\_NUMBER**
-

## tv\_trigger\_polarity

`niscope.Session.tv_trigger_polarity`

Specifies whether the video signal sync is positive or negative.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.VideoPolarity</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Video:Polarity**
  - C Attribute: **NISCOPE\_ATTR\_TV\_TRIGGER\_POLARITY**
- 

## tv\_trigger\_signal\_format

`niscope.Session.tv_trigger_signal_format`

Specifies the type of video signal, such as NTSC, PAL, or SECAM.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.VideoSignalFormat</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggering:Trigger Video:Signal Format**
  - C Attribute: **NISCOPE\_ATTR\_TV\_TRIGGER\_SIGNAL\_FORMAT**
- 

## use\_spec\_initial\_x

`niscope.Session.use_spec_initial_x`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>bool</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_USE\_SPEC\_INITIAL\_X**
- 

## vertical\_coupling

### `niscope.Session.vertical_coupling`

Specifies how the digitizer couples the input signal for the channel. When input coupling changes, the input stage takes a finite amount of time to settle.

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].vertical_coupling`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.vertical_coupling`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.VerticalCoupling</code>
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Vertical Coupling**
  - C Attribute: **NISCOPE\_ATTR\_VERTICAL\_COUPLING**
- 

## vertical\_offset

### `niscope.Session.vertical_offset`

Specifies the location of the center of the range. The value is with respect to ground and is in volts. For example, to acquire a sine wave that spans between 0.0 and 10.0 V, set this property to 5.0 V.

---

**Note:** This property is not supported by all digitizers. Refer to the NI High-Speed Digitizers Help for a list of vertical offsets supported for each device.

---

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].vertical_offset`

To set/get on all channels, you can call the property directly on the `niscope.Session`.



Example: `my_session.vertical_offset`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Vertical Offset**
  - C Attribute: **NISCOPE\_ATTR\_VERTICAL\_OFFSET**
- 

## vertical\_range

### `niscope.Session.vertical_range`

Specifies the absolute value of the input range for a channel in volts. For example, to acquire a sine wave that spans between -5 and +5 V, set this property to 10.0 V. Refer to the NI High-Speed Digitizers Help for a list of supported vertical ranges for each device. If the specified range is not supported by a device, the value is coerced up to the next valid range.

---

**Tip:** This property can be set/get on specific channels within your `niscope.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].vertical_range`

To set/get on all channels, you can call the property directly on the `niscope.Session`.

Example: `my_session.vertical_range`

---

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Vertical:Vertical Range**
  - C Attribute: **NISCOPE\_ATTR\_VERTICAL\_RANGE**
-

## width\_condition

### `niscope.Session.width_condition`

Specifies whether the oscilloscope triggers on pulses within or outside the duration range bounded by the `niscope.Session.width_low_threshold` and `niscope.Session.width_high_threshold` properties.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.WidthCondition</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: `NISCOPE_ATTR_WIDTH_CONDITION`
- 

## width\_high\_threshold

### `niscope.Session.width_high_threshold`

Specifies the high width threshold, in seconds.

This properties sets the upper bound on the duration range that triggers the oscilloscope. The `niscope.Session.width_condition` property determines how the oscilloscope triggers in relation to the width thresholds.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>float</code>
Permissions	read-write
Repeated Capabilities	None

---

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: `NISCOPE_ATTR_WIDTH_HIGH_THRESHOLD`
-

## width\_low\_threshold

### `niscope.Session.width_low_threshold`

Specifies the low width threshold, in seconds.

This property sets the lower bound on the duration range that triggers the oscilloscope. The [\*`niscope.Session.width\_condition`\*](#) property determines how the oscilloscope triggers in relation to the width thresholds.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_WIDTH\_LOW\_THRESHOLD**

## width\_polarity

### `niscope.Session.width_polarity`

Specifies the polarity of pulses that trigger the oscilloscope for width triggering.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	<code>enums.WidthPolarity</code>
Permissions	read-write
Repeated Capabilities	None

**Tip:** This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NISCOPE\_ATTR\_WIDTH\_POLARITY**

## NI-TClk Support

### `niscope.Session.tclk`

This is used to get and set NI-TClk attributes on the session.

**See also:**

See [`nitclk.SessionReference`](#) for a complete list of attributes.

Session

- *Session*
- *Methods*
  - *abort*
  - *acquisition\_status*
  - *add\_waveform\_processing*
  - *auto\_setup*
  - *clear\_waveform\_measurement\_stats*
  - *clear\_waveform\_processing*
  - *close*
  - *commit*
  - *configure\_chan\_characteristics*
  - *configure\_equalization\_filter\_coefficients*
  - *configure\_horizontal\_timing*
  - *configure\_trigger\_digital*
  - *configure\_trigger\_edge*
  - *configure\_trigger\_hysteresis*
  - *configure\_trigger\_immediate*
  - *configure\_trigger\_software*
  - *configure\_trigger\_video*
  - *configure\_trigger\_window*
  - *configure\_vertical*
  - *disable*
  - *export\_attribute\_configuration\_buffer*
  - *export\_attribute\_configuration\_file*
  - *fetch*
  - *fetch\_array\_measurement*
  - *fetch\_into*
  - *fetch\_measurement\_stats*
  - *get\_channel\_names*
  - *get\_equalization\_filter\_coefficients*
  - *get\_ext\_cal\_last\_date\_and\_time*
  - *get\_ext\_cal\_last\_temp*
  - *get\_self\_cal\_last\_date\_and\_time*
  - *get\_self\_cal\_last\_temp*
  - *import\_attribute\_configuration\_buffer*

- *import\_attribute\_configuration\_file*
- *initiate*
- *lock*
- *probe\_compensation\_signal\_start*
- *probe\_compensation\_signal\_stop*
- *read*
- *reset*
- *reset\_device*
- *reset\_with\_defaults*
- *self\_cal*
- *self\_test*
- *send\_software\_trigger\_edge*
- *unlock*
- *Properties*
  - *absolute\_sample\_clock\_offset*
  - *acquisition\_start\_time*
  - *acquisition\_type*
  - *acq\_arm\_source*
  - *advance\_trigger\_terminal\_name*
  - *adv\_trig\_src*
  - *allow\_more\_records\_than\_memory*
  - *arm\_ref\_trig\_src*
  - *backlog*
  - *bandpass\_filter\_enabled*
  - *binary\_sample\_width*
  - *cable\_sense\_mode*
  - *cable\_sense\_signal\_enable*
  - *cable\_sense\_voltage*
  - *channel\_count*
  - *channel\_enabled*
  - *channel\_terminal\_configuration*
  - *data\_transfer\_block\_size*
  - *data\_transfer\_maximum\_bandwidth*
  - *data\_transfer\_preferred\_packet\_size*
  - *device\_temperature*

- *enabled\_channels*
- *enable\_dc\_restore*
- *enable\_time\_interleaved\_sampling*
- *end\_of\_acquisition\_event\_output\_terminal*
- *end\_of\_acquisition\_event\_terminal\_name*
- *end\_of\_record\_event\_output\_terminal*
- *end\_of\_record\_event\_terminal\_name*
- *end\_of\_record\_to\_advance\_trigger\_holdoff*
- *equalization\_filter\_enabled*
- *equalization\_num\_coefficients*
- *exported\_advance\_trigger\_output\_terminal*
- *exported\_ref\_trigger\_output\_terminal*
- *exported\_start\_trigger\_output\_terminal*
- *flex\_fir\_antialias\_filter\_type*
- *fpga\_bitfile\_path*
- *glitch\_condition*
- *glitch\_polarity*
- *glitch\_width*
- *high\_pass\_filter\_frequency*
- *horz\_enforce\_realtime*
- *horz\_min\_num\_pts*
- *horz\_num\_records*
- *horz\_record\_length*
- *horz\_record\_ref\_position*
- *horz\_sample\_rate*
- *horz\_time\_per\_record*
- *input\_clock\_source*
- *input\_impedance*
- *instrument\_firmware\_revision*
- *instrument\_manufacturer*
- *instrument\_model*
- *interleaving\_offset\_correction\_enabled*
- *io\_resource\_descriptor*
- *is\_probe\_comp\_on*
- *logical\_name*

- *master\_enable*
- *max\_input\_frequency*
- *max\_real\_time\_sampling\_rate*
- *max\_ris\_rate*
- *meas\_array\_gain*
- *meas\_array\_offset*
- *meas\_chan\_high\_ref\_level*
- *meas\_chan\_low\_ref\_level*
- *meas\_chan\_mid\_ref\_level*
- *meas\_filter\_center\_freq*
- *meas\_filter\_cutoff\_freq*
- *meas\_filter\_order*
- *meas\_filter\_ripple*
- *meas\_filter\_taps*
- *meas\_filter\_transient\_waveform\_percent*
- *meas\_filter\_type*
- *meas\_filter\_width*
- *meas\_fir\_filter\_window*
- *meas\_high\_ref*
- *meas\_hysteresis\_percent*
- *meas\_interpolation\_sampling\_factor*
- *meas\_last\_acq\_histogram\_size*
- *meas\_low\_ref*
- *meas\_mid\_ref*
- *meas\_other\_channel*
- *meas\_percentage\_method*
- *meas\_polynomial\_interpolation\_order*
- *meas\_ref\_level\_units*
- *meas\_time\_histogram\_high\_time*
- *meas\_time\_histogram\_high\_volts*
- *meas\_time\_histogram\_low\_time*
- *meas\_time\_histogram\_low\_volts*
- *meas\_time\_histogram\_size*
- *meas\_voltage\_histogram\_high\_volts*
- *meas\_voltage\_histogram\_low\_volts*

- *meas\_voltage\_histogram\_size*
- *min\_sample\_rate*
- *onboard\_memory\_size*
- *output\_clock\_source*
- *pll\_lock\_status*
- *points\_done*
- *poll\_interval*
- *probe\_attenuation*
- *ready\_for\_advance\_event\_output\_terminal*
- *ready\_for\_advance\_event\_terminal\_name*
- *ready\_for\_ref\_event\_output\_terminal*
- *ready\_for\_ref\_event\_terminal\_name*
- *ready\_for\_start\_event\_output\_terminal*
- *ready\_for\_start\_event\_terminal\_name*
- *records\_done*
- *record\_arm\_source*
- *ref\_clk\_rate*
- *ref\_trigger\_detector\_location*
- *ref\_trigger\_minimum\_quiet\_time*
- *ref\_trigger\_terminal\_name*
- *ref\_trig\_tdc\_enable*
- *resolution*
- *ris\_in\_auto\_setup\_enable*
- *ris\_method*
- *ris\_num\_averages*
- *runt\_high\_threshold*
- *runt\_low\_threshold*
- *runt\_polarity*
- *runt\_time\_condition*
- *runt\_time\_high\_limit*
- *runt\_time\_low\_limit*
- *sample\_mode*
- *samp\_clk\_timebase\_div*
- *sample\_clock\_timebase\_multiplier*
- *samp\_clk\_timebase\_rate*



- *samp\_clk\_timebase\_src*
- *serial\_number*
- *accessory\_gain*
- *accessory\_offset*
- *simulate*
- *specific\_driver\_description*
- *specific\_driver\_revision*
- *specific\_driver\_vendor*
- *start\_to\_ref\_trigger\_holdoff*
- *start\_trigger\_terminal\_name*
- *supported\_instrument\_models*
- *trigger\_auto\_triggered*
- *trigger\_coupling*
- *trigger\_delay\_time*
- *trigger\_holdoff*
- *trigger\_hysteresis*
- *trigger\_impedance*
- *trigger\_level*
- *trigger\_modifier*
- *trigger\_slope*
- *trigger\_source*
- *trigger\_type*
- *trigger\_window\_high\_level*
- *trigger\_window\_low\_level*
- *trigger\_window\_mode*
- *tv\_trigger\_event*
- *tv\_trigger\_line\_number*
- *tv\_trigger\_polarity*
- *tv\_trigger\_signal\_format*
- *use\_spec\_initial\_x*
- *vertical\_coupling*
- *vertical\_offset*
- *vertical\_range*
- *width\_condition*
- *width\_high\_threshold*

- *width\_low\_threshold*
- *width\_polarity*
- *NI-TClk Support*

## Repeated Capabilities

Repeated capabilities attributes are used to set the *channel\_string* parameter to the underlying driver function call. This can be the actual function based on the `Session` method being called, or it can be the appropriate Get/Set Attribute function, such as `niScope_SetAttributeViInt32()`.

Repeated capabilities attributes use the indexing operator `[]` to indicate the repeated capabilities. The parameter can be a string, list, tuple, or slice (range). Each element of those can be a string or an integer. If it is a string, you can indicate a range using the same format as the driver: `'0-2'` or `'0:2'`

Some repeated capabilities use a prefix before the number and this is optional

## channels

`niscope.Session.channels`

```
session.channels['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

## instruments

`niscope.Session.instruments`

```
session.instruments['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

## Enums

Enums used in NI-SCOPE

## AcquisitionStatus

`class niscope.AcquisitionStatus`

`COMPLETE`

`IN_PROGRESS`

`STATUS_UNKNOWN`

## AcquisitionType

**class** niscopes.AcquisitionType

### **NORMAL**

Sets the digitizer to normal resolution mode. The digitizer can use real-time sampling or equivalent-time sampling.

### **FLEXRES**

Sets the digitizer to flexible resolution mode if supported. The digitizer uses different hardware configurations to change the resolution depending on the sampling rate used.

### **DDC**

Sets the digitizer to DDC mode on the NI 5620/5621.

## ArrayMeasurement

**class** niscopes.ArrayMeasurement

### **NO\_MEASUREMENT**

None

### **LAST\_ACQ\_HISTOGRAM**

Last Acquisition Histogram

### **FFT\_PHASE\_SPECTRUM**

FFT Phase Spectrum

### **FFT\_AMP\_SPECTRUM\_VOLTS\_RMS**

FFT Amp. Spectrum (Volts RMS)

### **MULTI\_ACQ\_VOLTAGE\_HISTOGRAM**

Multi Acquisition Voltage Histogram

### **MULTI\_ACQ\_TIME\_HISTOGRAM**

Multi Acquisition Time Histogram

### **ARRAY\_INTEGRAL**

Array Integral

### **DERIVATIVE**

Derivative

### **INVERSE**

Inverse

### **HANNING\_WINDOW**

Hanning Window

### **FLAT\_TOP\_WINDOW**

Flat Top Window

### **POLYNOMIAL\_INTERPOLATION**

Polynomial Interpolation

**MULTIPLY\_CHANNELS**

Multiply Channels

**ADD\_CHANNELS**

Add Channels

**SUBTRACT\_CHANNELS**

Subtract Channels

**DIVIDE\_CHANNELS**

Divide Channels

**MULTI\_ACQ\_AVERAGE**

Multi Acquisition Average

**BUTTERWORTH\_FILTER**

Butterworth IIR Filter

**CHEBYSHEV\_FILTER**

Chebyshev IIR Filter

**FFT\_AMP\_SPECTRUM\_DB**

FFT Amp. Spectrum (dB)

**HAMMING\_WINDOW**

Hamming Window

**WINDOWED\_FIR\_FILTER**

FIR Windowed Filter

**BESSEL\_FILTER**

Bessel IIR Filter

**TRIANGLE\_WINDOW**

Triangle Window

**BLACKMAN\_WINDOW**

Blackman Window

**ARRAY\_OFFSET**

Array Offset

**ARRAY\_GAIN**

Array Gain

## CableSenseMode

**class** nscope.CableSenseMode

**DISABLED**

The oscilloscope is not configured to emit a CableSense signal.

**ON\_DEMAND**

The oscilloscope is configured to emit a single CableSense pulse.

## ClearableMeasurement

```
class niscopes.ClearableMeasurement
```

```
    ALL_MEASUREMENTS
    MULTI_ACQ_VOLTAGE_HISTOGRAM
    MULTI_ACQ_TIME_HISTOGRAM
    MULTI_ACQ_AVERAGE
    FREQUENCY
    AVERAGE_FREQUENCY
    FFT_FREQUENCY
    PERIOD
    AVERAGE_PERIOD
    RISE_TIME
    FALL_TIME
    RISE_SLEW_RATE
    FALL_SLEW_RATE
    OVERSHOOT
    PRESHOOT
    VOLTAGE_RMS
    VOLTAGE_CYCLE_RMS
    AC_ESTIMATE
    FFT_AMPLITUDE
    VOLTAGE_AVERAGE
    VOLTAGE_CYCLE_AVERAGE
    DC_ESTIMATE
    VOLTAGE_MAX
    VOLTAGE_MIN
    VOLTAGE_PEAK_TO_PEAK
    VOLTAGE_HIGH
    VOLTAGE_LOW
    AMPLITUDE
```

VOLTAGE\_TOP  
 VOLTAGE\_BASE  
 VOLTAGE\_BASE\_TO\_TOP  
 WIDTH\_NEG  
 WIDTH\_POS  
 DUTY\_CYCLE\_NEG  
 DUTY\_CYCLE\_POS  
 INTEGRAL  
 AREA  
 CYCLE\_AREA  
 TIME\_DELAY  
 PHASE\_DELAY  
 LOW\_REF\_VOLTS  
 MID\_REF\_VOLTS  
 HIGH\_REF\_VOLTS  
 VOLTAGE\_HISTOGRAM\_MEAN  
 VOLTAGE\_HISTOGRAM\_STDEV  
 VOLTAGE\_HISTOGRAM\_MEDIAN  
 VOLTAGE\_HISTOGRAM\_MODE  
 VOLTAGE\_HISTOGRAM\_MAX  
 VOLTAGE\_HISTOGRAM\_MIN  
 VOLTAGE\_HISTOGRAM\_PEAK\_TO\_PEAK  
 VOLTAGE\_HISTOGRAM\_MEAN\_PLUS\_STDEV  
 VOLTAGE\_HISTOGRAM\_MEAN\_PLUS\_2\_STDEV  
 VOLTAGE\_HISTOGRAM\_MEAN\_PLUS\_3\_STDEV  
 VOLTAGE\_HISTOGRAM\_HITS  
 VOLTAGE\_HISTOGRAM\_NEW\_HITS  
 TIME\_HISTOGRAM\_MEAN  
 TIME\_HISTOGRAM\_STDEV  
 TIME\_HISTOGRAM\_MEDIAN  
 TIME\_HISTOGRAM\_MODE

**TIME\_HISTOGRAM\_MAX**  
**TIME\_HISTOGRAM\_MIN**  
**TIME\_HISTOGRAM\_PEAK\_TO\_PEAK**  
**TIME\_HISTOGRAM\_MEAN\_PLUS\_STDEV**  
**TIME\_HISTOGRAM\_MEAN\_PLUS\_2\_STDEV**  
**TIME\_HISTOGRAM\_MEAN\_PLUS\_3\_STDEV**  
**TIME\_HISTOGRAM\_HITS**  
**TIME\_HISTOGRAM\_NEW\_HITS**

## FIRFilterWindow

**class** niscopes.FIRFilterWindow

**NONE**  
No window.

**HANNING**  
Specifies a Hanning window.

**FLAT\_TOP**  
Specifies a Flat Top window.

**HAMMING**  
Specifies a Hamming window.

**TRIANGLE**  
Specifies a Triangle window.

**BLACKMAN**  
Specifies a Blackman window.

## FetchRelativeTo

**class** niscopes.FetchRelativeTo

**READ\_POINTER**  
The read pointer is set to zero when a new acquisition is initiated. After every fetch the read pointer is incremented to be the sample after the last sample retrieved. Therefore, you can repeatedly fetch relative to the read pointer for a continuous acquisition program.

**PRETRIGGER**  
Fetches relative to the first pretrigger point requested with `niscopes.Session.configure_horizontal_timing()`.

**NOW**  
Fetch data at the last sample acquired.

**START**

Fetch data starting at the first point sampled by the digitizer.

**TRIGGER**

Fetch at the first posttrigger sample.

## FilterType

**class** nscope.FilterType

**LOWPASS**

Specifies lowpass as the filter type.

**HIGHPASS**

Specifies highpass as the filter type.

**BANDPASS**

Specifies bandpass as the filter type.

**BANDSTOP**

Specifies bandstop as the filter type.

## FlexFIRAntialiasFilterType

**class** nscope.FlexFIRAntialiasFilterType

**FOURTYEIGHT\_TAP\_STANDARD**

This filter is optimized for alias protection and frequency-domain flatness

**FOURTYEIGHT\_TAP\_HANNING**

This filter is optimized for the lowest possible bandwidth for a 48 tap filter and maximizes the SNR

**SIXTEEN\_TAP\_HANNING**

This filter is optimized for the lowest possible bandwidth for a 16 tap filter and maximizes the SNR

**EIGHT\_TAP\_HANNING**

This filter is optimized for the lowest possible bandwidth for a 8 tap filter and maximizes the SNR

## GlitchCondition

**class** nscope.GlitchCondition

**GREATER**

Trigger on pulses with a duration greater than the specified glitch width.

**LESS**

Trigger on pulses with a duration shorter than the specified glitch width.



## GlitchPolarity

**class** nscope.GlitchPolarity

### POSITIVE

Trigger on pulses of positive polarity relative to the trigger threshold.

### NEGATIVE

Trigger on pulses of negative polarity relative to the trigger threshold.

### EITHER

Trigger on pulses of either positive or negative polarity.

## Option

**class** nscope.Option

### SELF\_CALIBRATE\_ALL\_CHANNELS

Self Calibrating all Channels

### RESTORE\_EXTERNAL\_CALIBRATION

Restore External Calibration.

## PercentageMethod

**class** nscope.PercentageMethod

### LOWHIGH

Specifies that the reference level percentages should be computed using the low/high method,

### MINMAX

Reference level percentages are computed using the min/max method.

### BASETOP

Reference level percentages are computed using the base/top method.

## RISMethod

**class** nscope.RISMethod

### EXACT\_NUM\_AVERAGES

Acquires exactly the specified number of records for each bin in the RIS acquisition. An error is returned from the fetch method if the RIS acquisition does not successfully acquire the specified number of waveforms within the timeout period. You may call the fetch method again to allow more time for the acquisition to finish.

### MIN\_NUM\_AVERAGES

Each RIS sample is the average of a least a minimum number of randomly distributed points.

**INCOMPLETE**

Returns the RIS waveform after the specified timeout even if it is incomplete. If no waveforms have been acquired in certain bins, these bins will have a NaN (when fetching scaled data) or a zero (when fetching binary data). A warning (positive error code) is returned from the fetch method if the RIS acquisition did not finish. The acquisition aborts when data is returned.

**LIMITED\_BIN\_WIDTH**

Limits the waveforms in the various bins to be within 200 ps of the center of the bin.

**RefLevelUnits**

**class** nscope.RefLevelUnits

**VOLTS**

Specifies that the reference levels are given in units of volts.

**PERCENTAGE**

(Default) Specifies that the reference levels are given in percentage units.

**RefTriggerDetectorLocation**

**class** nscope.RefTriggerDetectorLocation

**ANALOG\_DETECTION\_CIRCUIT**

use the hardware analog circuitry to implement the reference trigger. This option will trigger before any onboard signal processing.

**DDC\_OUTPUT**

use the onboard signal processing logic to implement the reference trigger. This option will trigger based on the onboard signal processed data.

**RuntPolarity**

**class** nscope.RuntPolarity

**POSITIVE**

Trigger on pulses of positive polarity relative to *nscope.Session.runt\_low\_threshold* that do not cross *nscope.Session.runt\_high\_threshold*.

**NEGATIVE**

Trigger on pulses of negative polarity relative to *nscope.Session.runt\_high\_threshold* that do not cross *nscope.Session.runt\_low\_threshold*.

**EITHER**

Trigger on pulses of either positive or negative polarity.

## RunTimeCondition

```
class nscope.RunTimeCondition
```

### NONE

Time qualification is disabled. Trigger on runt pulses based solely on the voltage level of the pulses.

### WITHIN

Trigger on pulses that, in addition to meeting runt voltage criteria, have a duration within the range bounded by *nscope.Session.runt\_time\_low\_limit* and *nscope.Session.runt\_time\_high\_limit*.

### OUTSIDE

Trigger on pulses that, in addition to meeting runt voltage criteria, have a duration not within the range bounded by *nscope.Session.runt\_time\_low\_limit* and *nscope.Session.runt\_time\_high\_limit*.

## ScalarMeasurement

```
class nscope.ScalarMeasurement
```

### NO\_MEASUREMENT

None

### RISE\_TIME

### FALL\_TIME

### FREQUENCY

### PERIOD

### VOLTAGE\_RMS

### VOLTAGE\_PEAK\_TO\_PEAK

### VOLTAGE\_MAX

### VOLTAGE\_MIN

### VOLTAGE\_HIGH

### VOLTAGE\_LOW

### VOLTAGE\_AVERAGE

### WIDTH\_NEG

### WIDTH\_POS

### DUTY\_CYCLE\_NEG

### DUTY\_CYCLE\_POS

### AMPLITUDE

### VOLTAGE\_CYCLE\_RMS

VOLTAGE\_CYCLE\_AVERAGE

OVERSHOOT

PRESHOOT

LOW\_REF\_VOLTS

MID\_REF\_VOLTS

HIGH\_REF\_VOLTS

AREA

CYCLE\_AREA

INTEGRAL

VOLTAGE\_BASE

VOLTAGE\_TOP

FFT\_FREQUENCY

FFT\_AMPLITUDE

RISE\_SLEW\_RATE

FALL\_SLEW\_RATE

AC\_ESTIMATE

DC\_ESTIMATE

TIME\_DELAY

AVERAGE\_PERIOD

AVERAGE\_FREQUENCY

VOLTAGE\_BASE\_TO\_TOP

PHASE\_DELAY

## TerminalConfiguration

```
class nscope.TerminalConfiguration
```

**SINGLE\_ENDED**

Channel is single ended

**UNBALANCED\_DIFFERENTIAL**

Channel is unbalanced differential

**DIFFERENTIAL**

Channel is differential

## TriggerCoupling

**class** nscope.TriggerCoupling

**AC**

AC coupling

**DC**

DC coupling

**HF\_REJECT**

Highpass filter coupling

**LF\_REJECT**

Lowpass filter coupling

**AC\_PLUS\_HF\_REJECT**

Highpass and lowpass filter coupling

## TriggerModifier

**class** nscope.TriggerModifier

**NO\_TRIGGER\_MOD**

Normal triggering.

**AUTO**

Software will trigger an acquisition automatically if no trigger arrives after a certain amount of time.

**AUTO\_LEVEL**

## TriggerSlope

**class** nscope.TriggerSlope

**NEGATIVE**

Falling edge

**POSITIVE**

Rising edge

**SLOPE\_EITHER**

Either edge

## TriggerType

**class** nscope.TriggerType

**EDGE**

Configures the digitizer for edge triggering. An edge trigger occurs when the trigger signal crosses the trigger level specified with the set trigger slope. You configure the trigger level and slope with *nscope.Session.configure\_trigger\_edge()*.

**HYSTERESIS**

Configures the digitizer for hysteresis triggering. A hysteresis trigger occurs when the trigger signal crosses the trigger level with the specified slope and passes through the hysteresis window you specify. You configure the trigger level, slope, and hysteresis with `niscope.Session.configure_trigger_hysteresis()`.

**DIGITAL**

Configures the digitizer for digital triggering. A digital trigger occurs when the trigger signal has the specified slope. You configure the trigger slope with `niscope.Session.configure_trigger_digital()`.

**WINDOW**

Configures the digitizer for window triggering. A window trigger occurs when the trigger signal enters or leaves the window defined by the values you specify with the Low Window Level, High Window Level, and Window Mode Parameters. You configure the low window level high window level, and window mode with `niscope.Session.configure_trigger_window()`.

**SOFTWARE**

Configures the digitizer for software triggering. A software trigger occurs when `niscope.Session.SendSoftwareTrigger()` is called.

**TV**

Configures the digitizer for video/TV triggering. You configure the video trigger parameters like signal Format, Line to trigger off of, Polarity, and Enable DC Restore with `niscope.Session.configure_trigger_video()`.

**GLITCH****WIDTH****RUNT****IMMEDIATE**

Configures the digitizer for immediate triggering. An immediate trigger occurs as soon as the pretrigger samples are acquired.

**TriggerWindowMode**

```
class niscope.TriggerWindowMode
```

**ENTERING**

Trigger upon entering the window

**LEAVING**

Trigger upon leaving the window

**ENTERING\_OR\_LEAVING**

## VerticalCoupling

**class** nscope.VerticalCoupling

**AC**

AC coupling

**DC**

DC coupling

**GND**

GND coupling

## VideoPolarity

**class** nscope.VideoPolarity

**POSITIVE**

Specifies that the video signal has positive polarity.

**NEGATIVE**

Specifies that the video signal has negative polarity.

## VideoSignalFormat

**class** nscope.VideoSignalFormat

**NTSC**

NTSC signal format supports line numbers from 1 to 525

**PAL**

PAL signal format supports line numbers from 1 to 625

**SECAM**

SECAM signal format supports line numbers from 1 to 625

**M\_PAL**

M-PAL signal format supports line numbers from 1 to 525

**VIDEO\_480I\_59\_94\_FIELDS\_PER\_SECOND**

480 lines, interlaced, 59.94 fields per second

**VIDEO\_480I\_60\_FIELDS\_PER\_SECOND**

480 lines, interlaced, 60 fields per second

**VIDEO\_480P\_59\_94\_FRAMES\_PER\_SECOND**

480 lines, progressive, 59.94 frames per second

**VIDEO\_480P\_60\_FRAMES\_PER\_SECOND**

480 lines, progressive, 60 frames per second

**VIDEO\_576I\_50\_FIELDS\_PER\_SECOND**

576 lines, interlaced, 50 fields per second

**VIDEO\_576P\_50\_FRAMES\_PER\_SECOND**

576 lines, progressive, 50 frames per second

**VIDEO\_720P\_50\_FRAMES\_PER\_SECOND**

720 lines, progressive, 50 frames per second

**VIDEO\_720P\_59\_94\_FRAMES\_PER\_SECOND**

720 lines, progressive, 59.94 frames per second

**VIDEO\_720P\_60\_FRAMES\_PER\_SECOND**

720 lines, progressive, 60 frames per second

**VIDEO\_1080I\_50\_FIELDS\_PER\_SECOND**

1,080 lines, interlaced, 50 fields per second

**VIDEO\_1080I\_59\_94\_FIELDS\_PER\_SECOND**

1,080 lines, interlaced, 59.94 fields per second

**VIDEO\_1080I\_60\_FIELDS\_PER\_SECOND**

1,080 lines, interlaced, 60 fields per second

**VIDEO\_1080P\_24\_FRAMES\_PER\_SECOND**

1,080 lines, progressive, 24 frames per second

## VideoTriggerEvent

```
class nscope.VideoTriggerEvent
```

**FIELD1**

Trigger on field 1 of the signal

**FIELD2**

Trigger on field 2 of the signal

**ANY\_FIELD**

Trigger on the first field acquired

**ANY\_LINE**

Trigger on the first line acquired

**LINE\_NUMBER**

Trigger on a specific line of a video signal. Valid values vary depending on the signal format configured.

## WhichTrigger

```
class nscope.WhichTrigger
```

**START****ARM\_REFERENCE****REFERENCE****ADVANCE**



## WidthCondition

**class** niscopes.WidthCondition

### WITHIN

Trigger on pulses with a duration within the range bounded by *niscopes.Session.width\_low\_threshold* and *niscopes.Session.width\_high\_threshold*.

### OUTSIDE

Trigger on pulses with a duration not within the range bounded by *niscopes.Session.width\_low\_threshold* and *niscopes.Session.width\_high\_threshold*.

## WidthPolarity

**class** niscopes.WidthPolarity

### POSITIVE

Trigger on pulses of positive polarity relative to the trigger threshold.

### NEGATIVE

Trigger on pulses of negative polarity relative to the trigger threshold.

### EITHER

Trigger on pulses of either positive or negative polarity.

## Exceptions and Warnings

### Error

**exception** niscopes.errors.Error

Base exception type that all NI-SCOPE exceptions derive from

### DriverError

**exception** niscopes.errors.DriverError

An error originating from the NI-SCOPE driver

### UnsupportedConfigurationError

**exception** niscopes.errors.UnsupportedConfigurationError

An error due to using this module in an unsupported platform.

### DriverNotInstalledError

**exception** `niscope.errors.DriverNotInstalledError`

An error due to using this module without the driver runtime installed.

### DriverTooOldError

**exception** `niscope.errors.DriverTooOldError`

An error due to using this module with an older version of the NI-SCOPE driver runtime.

### DriverTooNewError

**exception** `niscope.errors.DriverTooNewError`

An error due to the NI-SCOPE driver runtime being too new for this module.

### InvalidRepeatedCapabilityError

**exception** `niscope.errors.InvalidRepeatedCapabilityError`

An error due to an invalid character in a repeated capability

### SelfTestError

**exception** `niscope.errors.SelfTestError`

An error due to a failed self-test

### RpcError

**exception** `niscope.errors.RpcError`

An error specific to sessions to the NI gRPC Device Server

### DriverWarning

**exception** `niscope.errors.DriverWarning`

A warning originating from the NI-SCOPE driver

### Examples

You can download all [niscope examples here](#)

## niscope\_fetch.py

Listing 1: (niscope\_fetch.py)

```

1  #!/usr/bin/python
2
3  import argparse
4  import niscopes
5  import pprint
6  import sys
7
8  pp = pprint.PrettyPrinter(indent=4, width=80)
9
10
11 def example(resource_name, channels, options, length, voltage):
12     with niscopes.Session(resource_name=resource_name, options=options) as session:
13         session.configure_vertical(range=voltage, coupling=niscopes.VerticalCoupling.AC)
14         session.configure_horizontal_timing(min_sample_rate=50000000, min_num_pts=length,
15     ↪ ref_position=50.0, num_records=1, enforce_realtime=True)
16         with session.initiate():
17             waveforms = session.channels[channels].fetch(num_samples=length)
18             for i in range(len(waveforms)):
19                 print(f'Waveform {i} information:')
20                 print(str(waveforms[i]) + '\n\n')
21
22 def _main(argv):
23     parser = argparse.ArgumentParser(description='Acquires one record from the given
24     ↪ channels.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
25     parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource
26     ↪ name of an NI digitizer.')
27     parser.add_argument('-c', '--channels', default='0', help='Channel(s) to use')
28     parser.add_argument('-l', '--length', default=1000, type=int, help='Measure record
29     ↪ length')
30     parser.add_argument('-v', '--voltage', default=1.0, type=float, help='Voltage range
31     ↪ (V)')
32     parser.add_argument('-op', '--option-string', default='', type=str, help='Option
33     ↪ string')
34     args = parser.parse_args(argv)
35     example(args.resource_name, args.channels, args.option_string, args.length, args
36     ↪ voltage)
37
38 def main():
39     _main(sys.argv[1:])
40
41 def test_example():
42     options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe', }
43     ↪ , }
44     example('PXI1Slot2', '0', options, 1000, 1.0)

```

(continues on next page)

(continued from previous page)

```

42 def test_main():
43     cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe',
44     ↪ ]
45     _main(cmd_line)
46
47 if __name__ == '__main__':
48     main()
49

```

**niscope\_fetch\_forever.py**

Listing 2: (niscope\_fetch\_forever.py)

```

1  #!/usr/bin/python
2
3  import argparse
4  import hightime
5  import niscopes
6  import numpy as np
7  import pprint
8  import sys
9
10
11 pp = pprint.PrettyPrinter(indent=4, width=80)
12
13
14 # We use fetch_into which allows us to allocate a single buffer per channel and "fetch_
15 ↪ into" it a section at a time without having to
16 # reconstruct the waveform once we are done
17 def example(resource_name, options, total_acquisition_time_in_seconds, voltage, sample_
18 ↪ rate_in_hz, samples_per_fetch):
19     total_samples = int(total_acquisition_time_in_seconds * sample_rate_in_hz)
20     # 1. Opening session
21     with niscopes.Session(resource_name=resource_name, options=options) as session:
22         # We will acquire on all channels of the device
23         channel_list = [c for c in range(session.channel_count)] # Need an actual list_
24 ↪ and not a range
25
26         # 2. Creating numpy arrays
27         waveforms = [np.ndarray(total_samples, dtype=np.float64) for c in channel_list]
28
29         # 3. Configuring
30         session.configure_horizontal_timing(min_sample_rate=sample_rate_in_hz, min_num_
31 ↪ pts=1, ref_position=0.0, num_records=1, enforce_realtime=True)
32         session.channels[channel_list].configure_vertical(voltage, coupling=niscopes.
33 ↪ VerticalCoupling.DC, enabled=True)
34         # Configure software trigger, but never send the trigger.
35         # This starts an infinite acquisition, until you call session.abort() or session.
36 ↪ close()

```

(continues on next page)

(continued from previous page)

```

31     session.configure_trigger_software()
32     current_pos = 0
33     # 4. initiating
34     with session.initiate():
35         while current_pos < total_samples:
36             # We fetch each channel at a time so we don't have to de-interleave
37             ↪ afterwards
38             # We do not keep the wfm_info returned from fetch_into
39             for channel, waveform in zip(channel_list, waveforms):
40                 # 5. fetching - we return the slice of the waveform array that we
41                 ↪ want to "fetch into"
42                 session.channels[channel].fetch_into(waveform[current_pos:current_
43                 ↪ pos + samples_per_fetch], relative_to=niscope.FetchRelativeTo.READ_POINTER,
44                 ↪ offset=0, record_number=0, num_
45                 ↪ records=1, timeout=hightime.timedelta(seconds=5.0))
46                 current_pos += samples_per_fetch
47
48 def _main(argv):
49     parser = argparse.ArgumentParser(description='Fetch more samples than will fit in
50     ↪ memory.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
51     parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource
52     ↪ name of an NI digitizer.')
53     parser.add_argument('-t', '--time', default=10, type=int, help='Time to sample (s)')
54     parser.add_argument('-v', '--voltage', default=1.0, type=float, help='Voltage range
55     ↪ (V)')
56     parser.add_argument('-op', '--option-string', default='', type=str, help='Option
57     ↪ string')
58     parser.add_argument('-r', '--sample-rate', default=1000.0, type=float, help='Sample
59     ↪ Rate (Hz)')
60     parser.add_argument('-s', '--samples-per-fetch', default=100, type=int, help=
61     ↪ 'Samples per fetch')
62     args = parser.parse_args(argv)
63     example(args.resource_name, args.option_string, args.time, args.voltage, args.sample_
64     ↪ rate, args.samples_per_fetch)
65
66 def main():
67     _main(sys.argv[1:])
68
69 def test_example():
70     options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe', }
71     ↪ , }
72     example('PXI1Slot2', options, 10, 1.0, 1000.0, 100)
73
74 def test_main():
75     cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe',
76     ↪ ]
77     _main(cmd_line)

```

(continues on next page)

(continued from previous page)

```

70
71 if __name__ == '__main__':
72     main()
73

```

## niscope\_fetch\_into.py

Listing 3: (niscope\_fetch\_into.py)

```

1  #!/usr/bin/python
2
3  import argparse
4  import niscope
5  import numpy
6  import pprint
7  import sys
8
9  pp = pprint.PrettyPrinter(indent=4, width=80)
10
11
12 def example(resource_name, channels, options, length, voltage):
13     # fetch_into() allows you to preallocate and reuse the destination of the fetched_
14     ↪ waveforms, which can result in better performance at the expense of the usability of_
15     ↪ fetch().
16     channels = [ch.strip() for ch in channels.split(",")]
17     num_channels = len(channels)
18     num_records = 5
19     total_num_wfms = num_channels * num_records
20     # preallocate a single array for all samples in all waveforms
21     # Supported array types are: numpy.float64, numpy.int8, numpy.int16, numpy.int32
22     # int8, int16, int32 are for fetching unscaled data, which is the fastest way to_
23     ↪ fetch.
24     # Gain and Offset are stored in the returned WaveformInfo objects and can be applied_
25     ↪ to the data by the user later.
26     wfm = numpy.ndarray(length * total_num_wfms, dtype=numpy.float64)
27     with niscope.Session(resource_name=resource_name, options=options) as session:
28         session.configure_vertical(range=voltage, coupling=niscope.VerticalCoupling.AC)
29         session.configure_horizontal_timing(min_sample_rate=50000000, min_num_pts=length,
30         ↪ ref_position=50.0, num_records=num_records, enforce_realtime=True)
31         with session.initiate():
32             waveforms = session.channels[channels].fetch_into(waveform=wfm, num_
33             ↪ records=num_records)
34             for i in range(len(waveforms)):
35                 print(f'Waveform {i} information:')
36                 print(f'{waveforms[i]}\n\n')
37                 print(f'Samples: {waveforms[i].samples.tolist()}')
38
39 def _main(argv):
40     parser = argparse.ArgumentParser(description='Fetches data directly into a_

```

(continues on next page)

(continued from previous page)

```

36     ↪preallocated numpy array.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource
37     ↪name of an NI digitizer.')
    parser.add_argument('-c', '--channels', default='0', help='Channel(s) to use')
38     parser.add_argument('-l', '--length', default=100, type=int, help='Measure record
    ↪length')
39     parser.add_argument('-v', '--voltage', default=1.0, type=float, help='Voltage range
    ↪(V)')
40     parser.add_argument('-op', '--option-string', default='', type=str, help='Option
    ↪string')
41     args = parser.parse_args(argv)
42     example(args.resource_name, args.channels, args.option_string, args.length, args.
    ↪voltage)
43
44
45 def main():
46     _main(sys.argv[1:])
47
48
49 def test_example():
50     options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe', }
    ↪, }
51     example('PXI1Slot2', '0', 1, options, 100, 1.0)
52
53
54 def test_main():
55     cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe',
    ↪]
56     _main(cmd_line)
57
58
59 if __name__ == '__main__':
60     main()
61

```

## niscope\_read.py

Listing 4: (niscope\_read.py)

```

1  #!/usr/bin/python
2
3  import argparse
4  import niscope
5  import pprint
6  import sys
7
8  pp = pprint.PrettyPrinter(indent=4, width=80)
9
10
11 def example(resource_name, channels, options, length, voltage):

```

(continues on next page)

(continued from previous page)

```

12     with niscopes.Session(resource_name=resource_name, options=options) as session:
13         session.configure_vertical(range=voltage, coupling=niscopes.VerticalCoupling.AC)
14         session.configure_horizontal_timing(min_sample_rate=50000000, min_num_pts=length,
↪ ref_position=50.0, num_records=1, enforce_realtime=True)
15         waveforms = session.channels[channels].read(num_samples=length)
16         for i in range(len(waveforms)):
17             print(f'Waveform {i} information:')
18             print(str(waveforms[i]) + '\n\n')
19
20
21 def _main(argv):
22     parser = argparse.ArgumentParser(description='Acquires one record from the given
↪ channels.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
23     parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource
↪ name of an NI digitizer.')
24     parser.add_argument('-c', '--channels', default='0', help='Channel(s) to use')
25     parser.add_argument('-l', '--length', default=1000, type=int, help='Measure record
↪ length')
26     parser.add_argument('-v', '--voltage', default=1.0, type=float, help='Voltage range
↪ (V)')
27     parser.add_argument('-op', '--option-string', default='', type=str, help='Option
↪ string')
28     args = parser.parse_args(argv)
29     example(args.resource_name, args.channels, args.option_string, args.length, args.
↪ voltage)
30
31
32 def main():
33     _main(sys.argv[1:])
34
35
36 def test_example():
37     options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe', }
↪ , }
38     example('PXI1Slot2', '0', options, 1000, 1.0)
39
40
41 def test_main():
42     cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe',
↪ ]
43     _main(cmd_line)
44
45
46 if __name__ == '__main__':
47     main()
48

```



## gRPC Support

Support for using NI-SCOPE over gRPC

### SessionInitializationBehavior

**class** nscope.SessionInitializationBehavior

#### AUTO

The NI gRPC Device Server will attach to an existing session with the specified name if it exists, otherwise the server will initialize a new session.

---

**Note:** When using the Session as a context manager and the context exits, the behavior depends on what happened when the constructor was called. If it resulted in a new session being initialized on the NI gRPC Device Server, then it will automatically close the server session. If it instead attached to an existing session, then it will detach from the server session and leave it open.

---

#### INITIALIZE\_SERVER\_SESSION

Require the NI gRPC Device Server to initialize a new session with the specified name.

---

**Note:** When using the Session as a context manager and the context exits, it will automatically close the server session.

---

#### ATTACH\_TO\_SERVER\_SESSION

Require the NI gRPC Device Server to attach to an existing session with the specified name.

---

**Note:** When using the Session as a context manager and the context exits, it will detach from the server session and leave it open.

---

## GrpcSessionOptions

**class** nscope.GrpcSessionOptions(*self*, *grpc\_channel*, *session\_name*,  
                                  *initialization\_behavior*=SessionInitializationBehavior.AUTO)

Collection of options that specifies session behaviors related to gRPC.

Creates and returns an object you can pass to a Session constructor.

#### Parameters

- **grpc\_channel** (*grpc.Channel*) – Specifies the channel to the NI gRPC Device Server.
- **session\_name** (*str*) – User-specified name that identifies the driver session on the NI gRPC Device Server.

This is different from the resource name parameter many APIs take as a separate parameter. Specifying a name makes it easy to share sessions across multiple gRPC clients. You can use an empty string if you want to always initialize a new session on the server. To attach to an existing session, you must specify the session name it was initialized with.

- **initialization\_behavior** (*[niscopes.SessionInitializationBehavior](#)*) – Specifies whether it is acceptable to initialize a new session or attach to an existing one, or if only one of the behaviors is desired.

The driver session exists on the NI gRPC Device Server.

## 4.2 Additional Documentation

Refer to your driver documentation for device-specific information and detailed API documentation.

Refer to the [nimi-python Read the Docs project](#) for documentation of versions 1.4.4 of the module or earlier.

## **LICENSE**

**nimi-python** is licensed under an MIT-style license (see [LICENSE](#)). Other incorporated projects may be licensed under different licenses. All licenses allow for non-commercial and commercial use.

### **gRPC Features**

For driver APIs that support it, passing a `GrpcSessionOptions` instance as a parameter to `Session.__init__()` is subject to the NI General Purpose EULA (see [NILICENSE](#)).



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

n

niscope, 8





## A

`abort()` (in module *niscopes.Session*), 10  
`absolute_sample_clock_offset` (in module *niscopes.Session*), 36  
`AC` (*niscopes.TriggerCoupling* attribute), 137  
`AC` (*niscopes.VerticalCoupling* attribute), 139  
`AC_ESTIMATE` (*niscopes.ClearableMeasurement* attribute), 129  
`AC_ESTIMATE` (*niscopes.ScalarMeasurement* attribute), 136  
`AC_PLUS_HF_REJECT` (*niscopes.TriggerCoupling* attribute), 137  
`accessory_gain` (in module *niscopes.Session*), 102  
`accessory_offset` (in module *niscopes.Session*), 103  
`acq_arm_source` (in module *niscopes.Session*), 38  
`acquisition_start_time` (in module *niscopes.Session*), 37  
`acquisition_status()` (in module *niscopes.Session*), 10  
`acquisition_type` (in module *niscopes.Session*), 37  
`AcquisitionStatus` (class in *niscopes*), 126  
`AcquisitionType` (class in *niscopes*), 127  
`ADD_CHANNELS` (*niscopes.ArrayMeasurement* attribute), 128  
`add_waveform_processing()` (in module *niscopes.Session*), 11  
`adv_trig_src` (in module *niscopes.Session*), 39  
`ADVANCE` (*niscopes.WhichTrigger* attribute), 140  
`advance_trigger_terminal_name` (in module *niscopes.Session*), 38  
`ALL_MEASUREMENTS` (*niscopes.ClearableMeasurement* attribute), 129  
`allow_more_records_than_memory` (in module *niscopes.Session*), 39  
`AMPLITUDE` (*niscopes.ClearableMeasurement* attribute), 129  
`AMPLITUDE` (*niscopes.ScalarMeasurement* attribute), 135  
`ANALOG_DETECTION_CIRCUIT` (*niscopes.RefTriggerDetectorLocation* attribute), 134  
`ANY_FIELD` (*niscopes.VideoTriggerEvent* attribute), 140  
`ANY_LINE` (*niscopes.VideoTriggerEvent* attribute), 140

`AREA` (*niscopes.ClearableMeasurement* attribute), 130  
`AREA` (*niscopes.ScalarMeasurement* attribute), 136  
`arm_ref_trig_src` (in module *niscopes.Session*), 40  
`ARM_REFERENCE` (*niscopes.WhichTrigger* attribute), 140  
`ARRAY_GAIN` (*niscopes.ArrayMeasurement* attribute), 128  
`ARRAY_INTEGRAL` (*niscopes.ArrayMeasurement* attribute), 127  
`ARRAY_OFFSET` (*niscopes.ArrayMeasurement* attribute), 128  
`ArrayMeasurement` (class in *niscopes*), 127  
`ATTACH_TO_SERVER_SESSION` (*niscopes.SessionInitializationBehavior* attribute), 149  
`AUTO` (*niscopes.SessionInitializationBehavior* attribute), 149  
`AUTO` (*niscopes.TriggerModifier* attribute), 137  
`AUTO_LEVEL` (*niscopes.TriggerModifier* attribute), 137  
`auto_setup()` (in module *niscopes.Session*), 11  
`AVERAGE_FREQUENCY` (*niscopes.ClearableMeasurement* attribute), 129  
`AVERAGE_FREQUENCY` (*niscopes.ScalarMeasurement* attribute), 136  
`AVERAGE_PERIOD` (*niscopes.ClearableMeasurement* attribute), 129  
`AVERAGE_PERIOD` (*niscopes.ScalarMeasurement* attribute), 136

## B

`backlog` (in module *niscopes.Session*), 40  
`BANDPASS` (*niscopes.FilterType* attribute), 132  
`bandpass_filter_enabled` (in module *niscopes.Session*), 41  
`BANDSTOP` (*niscopes.FilterType* attribute), 132  
`BASETOP` (*niscopes.PercentageMethod* attribute), 133  
`BESSEL_FILTER` (*niscopes.ArrayMeasurement* attribute), 128  
`binary_sample_width` (in module *niscopes.Session*), 41  
`BLACKMAN` (*niscopes.FIRFilterWindow* attribute), 131  
`BLACKMAN_WINDOW` (*niscopes.ArrayMeasurement* attribute), 128  
`BUTTERWORTH_FILTER` (*niscopes.ArrayMeasurement* attribute), 128

## C

[cable\\_sense\\_mode](#) (in module [niscope.Session](#)), 42  
[cable\\_sense\\_signal\\_enable](#) (in module [niscope.Session](#)), 43  
[cable\\_sense\\_voltage](#) (in module [niscope.Session](#)), 43  
[CableSenseMode](#) (class in [niscope](#)), 128  
[channel\\_count](#) (in module [niscope.Session](#)), 44  
[channel\\_enabled](#) (in module [niscope.Session](#)), 44  
[channel\\_terminal\\_configuration](#) (in module [niscope.Session](#)), 45  
[channels](#) ([niscope.Session.niscope.Session](#) attribute), 126  
[CHEBYSHEV\\_FILTER](#) ([niscope.ArrayMeasurement](#) attribute), 128  
[clear\\_waveform\\_measurement\\_stats\(\)](#) (in module [niscope.Session](#)), 12  
[clear\\_waveform\\_processing\(\)](#) (in module [niscope.Session](#)), 13  
[ClearableMeasurement](#) (class in [niscope](#)), 129  
[close\(\)](#) (in module [niscope.Session](#)), 13  
[commit\(\)](#) (in module [niscope.Session](#)), 13  
[COMPLETE](#) ([niscope.AcquisitionStatus](#) attribute), 126  
[configure\\_chan\\_characteristics\(\)](#) (in module [niscope.Session](#)), 13  
[configure\\_equalization\\_filter\\_coefficients\(\)](#) (in module [niscope.Session](#)), 14  
[configure\\_horizontal\\_timing\(\)](#) (in module [niscope.Session](#)), 14  
[configure\\_trigger\\_digital\(\)](#) (in module [niscope.Session](#)), 15  
[configure\\_trigger\\_edge\(\)](#) (in module [niscope.Session](#)), 16  
[configure\\_trigger\\_hysteresis\(\)](#) (in module [niscope.Session](#)), 17  
[configure\\_trigger\\_immediate\(\)](#) (in module [niscope.Session](#)), 18  
[configure\\_trigger\\_software\(\)](#) (in module [niscope.Session](#)), 18  
[configure\\_trigger\\_video\(\)](#) (in module [niscope.Session](#)), 19  
[configure\\_trigger\\_window\(\)](#) (in module [niscope.Session](#)), 20  
[configure\\_vertical\(\)](#) (in module [niscope.Session](#)), 21  
[CYCLE\\_AREA](#) ([niscope.ClearableMeasurement](#) attribute), 130  
[CYCLE\\_AREA](#) ([niscope.ScalarMeasurement](#) attribute), 136

## D

[data\\_transfer\\_block\\_size](#) (in module [niscope.Session](#)), 45  
[data\\_transfer\\_maximum\\_bandwidth](#) (in module [niscope.Session](#)), 46

[data\\_transfer\\_preferred\\_packet\\_size](#) (in module [niscope.Session](#)), 46  
[DC](#) ([niscope.TriggerCoupling](#) attribute), 137  
[DC](#) ([niscope.VerticalCoupling](#) attribute), 139  
[DC\\_ESTIMATE](#) ([niscope.ClearableMeasurement](#) attribute), 129  
[DC\\_ESTIMATE](#) ([niscope.ScalarMeasurement](#) attribute), 136  
[DDC](#) ([niscope.AcquisitionType](#) attribute), 127  
[DDC\\_OUTPUT](#) ([niscope.RefTriggerDetectorLocation](#) attribute), 134  
[DERIVATIVE](#) ([niscope.ArrayMeasurement](#) attribute), 127  
[device\\_temperature](#) (in module [niscope.Session](#)), 46  
[DIFFERENTIAL](#) ([niscope.TerminalConfiguration](#) attribute), 136  
[DIGITAL](#) ([niscope.TriggerType](#) attribute), 138  
[disable\(\)](#) (in module [niscope.Session](#)), 21  
[DISABLED](#) ([niscope.CableSenseMode](#) attribute), 128  
[DIVIDE\\_CHANNELS](#) ([niscope.ArrayMeasurement](#) attribute), 128  
[DriverError](#), 141  
[DriverNotInstalledError](#), 142  
[DriverTooNewError](#), 142  
[DriverTooOldError](#), 142  
[DriverWarning](#), 142  
[DUTY\\_CYCLE\\_NEG](#) ([niscope.ClearableMeasurement](#) attribute), 130  
[DUTY\\_CYCLE\\_NEG](#) ([niscope.ScalarMeasurement](#) attribute), 135  
[DUTY\\_CYCLE\\_POS](#) ([niscope.ClearableMeasurement](#) attribute), 130  
[DUTY\\_CYCLE\\_POS](#) ([niscope.ScalarMeasurement](#) attribute), 135

## E

[EDGE](#) ([niscope.TriggerType](#) attribute), 137  
[EIGHT\\_TAP\\_HANNING](#) ([niscope.FlexFIRAntialiasFilterType](#) attribute), 132  
[EITHER](#) ([niscope.GlitchPolarity](#) attribute), 133  
[EITHER](#) ([niscope.RuntPolarity](#) attribute), 134  
[EITHER](#) ([niscope.WidthPolarity](#) attribute), 141  
[enable\\_dc\\_restore](#) (in module [niscope.Session](#)), 48  
[enable\\_time\\_interleaved\\_sampling](#) (in module [niscope.Session](#)), 48  
[enabled\\_channels](#) (in module [niscope.Session](#)), 47  
[end\\_of\\_acquisition\\_event\\_output\\_terminal](#) (in module [niscope.Session](#)), 49  
[end\\_of\\_acquisition\\_event\\_terminal\\_name](#) (in module [niscope.Session](#)), 49  
[end\\_of\\_record\\_event\\_output\\_terminal](#) (in module [niscope.Session](#)), 50  
[end\\_of\\_record\\_event\\_terminal\\_name](#) (in module [niscope.Session](#)), 50

- end\_of\_record\_to\_advance\_trigger\_holdoff (in module *niscope.Session*), 51
- ENTERING (*niscope.TriggerWindowMode* attribute), 138
- ENTERING\_OR\_LEAVING (*niscope.TriggerWindowMode* attribute), 138
- equalization\_filter\_enabled (in module *niscope.Session*), 51
- equalization\_num\_coefficients (in module *niscope.Session*), 52
- Error, 141
- EXACT\_NUM\_AVERAGES (*niscope.RISMethod* attribute), 133
- export\_attribute\_configuration\_buffer() (in module *niscope.Session*), 21
- export\_attribute\_configuration\_file() (in module *niscope.Session*), 22
- exported\_advance\_trigger\_output\_terminal (in module *niscope.Session*), 52
- exported\_ref\_trigger\_output\_terminal (in module *niscope.Session*), 53
- exported\_start\_trigger\_output\_terminal (in module *niscope.Session*), 53
- ## F
- FALL\_SLEW\_RATE (*niscope.ClearableMeasurement* attribute), 129
- FALL\_SLEW\_RATE (*niscope.ScalarMeasurement* attribute), 136
- FALL\_TIME (*niscope.ClearableMeasurement* attribute), 129
- FALL\_TIME (*niscope.ScalarMeasurement* attribute), 135
- fetch() (in module *niscope.Session*), 22
- fetch\_array\_measurement() (in module *niscope.Session*), 24
- fetch\_into() (in module *niscope.Session*), 25
- fetch\_measurement\_stats() (in module *niscope.Session*), 27
- FetchRelativeTo (class in *niscope*), 131
- FFT\_AMP\_SPECTRUM\_DB (*niscope.ArrayMeasurement* attribute), 128
- FFT\_AMP\_SPECTRUM\_VOLTS\_RMS (*niscope.ArrayMeasurement* attribute), 127
- FFT\_AMPLITUDE (*niscope.ClearableMeasurement* attribute), 129
- FFT\_AMPLITUDE (*niscope.ScalarMeasurement* attribute), 136
- FFT\_FREQUENCY (*niscope.ClearableMeasurement* attribute), 129
- FFT\_FREQUENCY (*niscope.ScalarMeasurement* attribute), 136
- FFT\_PHASE\_SPECTRUM (*niscope.ArrayMeasurement* attribute), 127
- FIELD1 (*niscope.VideoTriggerEvent* attribute), 140
- FIELD2 (*niscope.VideoTriggerEvent* attribute), 140
- FilterType (class in *niscope*), 132
- FIRFilterWindow (class in *niscope*), 131
- FLAT\_TOP (*niscope.FIRFilterWindow* attribute), 131
- FLAT\_TOP\_WINDOW (*niscope.ArrayMeasurement* attribute), 127
- flex\_fir\_antialias\_filter\_type (in module *niscope.Session*), 54
- FlexFIRAntialiasFilterType (class in *niscope*), 132
- FLEXRES (*niscope.AcquisitionType* attribute), 127
- FOURTYEIGHT\_TAP\_HANNING (*niscope.FlexFIRAntialiasFilterType* attribute), 132
- FOURTYEIGHT\_TAP\_STANDARD (*niscope.FlexFIRAntialiasFilterType* attribute), 132
- fpga\_bitfile\_path (in module *niscope.Session*), 54
- FREQUENCY (*niscope.ClearableMeasurement* attribute), 129
- FREQUENCY (*niscope.ScalarMeasurement* attribute), 135
- ## G
- get\_channel\_names() (in module *niscope.Session*), 28
- get\_equalization\_filter\_coefficients() (in module *niscope.Session*), 29
- get\_ext\_cal\_last\_date\_and\_time() (in module *niscope.Session*), 29
- get\_ext\_cal\_last\_temp() (in module *niscope.Session*), 29
- get\_self\_cal\_last\_date\_and\_time() (in module *niscope.Session*), 30
- get\_self\_cal\_last\_temp() (in module *niscope.Session*), 30
- GLITCH (*niscope.TriggerType* attribute), 138
- glitch\_condition (in module *niscope.Session*), 55
- glitch\_polarity (in module *niscope.Session*), 55
- glitch\_width (in module *niscope.Session*), 55
- GlitchCondition (class in *niscope*), 132
- GlitchPolarity (class in *niscope*), 133
- GND (*niscope.VerticalCoupling* attribute), 139
- GREATER (*niscope.GlitchCondition* attribute), 132
- GrpcSessionOptions (class in *niscope*), 149
- ## H
- HAMMING (*niscope.FIRFilterWindow* attribute), 131
- HAMMING\_WINDOW (*niscope.ArrayMeasurement* attribute), 128
- HANNING (*niscope.FIRFilterWindow* attribute), 131
- HANNING\_WINDOW (*niscope.ArrayMeasurement* attribute), 127
- HF\_REJECT (*niscope.TriggerCoupling* attribute), 137
- high\_pass\_filter\_frequency (in module *niscope.Session*), 56
- HIGH\_REF\_VOLTS (*niscope.ClearableMeasurement* attribute), 130

HIGH\_REF\_VOLTS (*niscopes.ScalarMeasurement* attribute), 136  
HIGHPASS (*niscopes.FilterType* attribute), 132  
horz\_enforce\_realtime (in module *niscopes.Session*), 56  
horz\_min\_num\_pts (in module *niscopes.Session*), 57  
horz\_num\_records (in module *niscopes.Session*), 57  
horz\_record\_length (in module *niscopes.Session*), 58  
horz\_record\_ref\_position (in module *niscopes.Session*), 58  
horz\_sample\_rate (in module *niscopes.Session*), 59  
horz\_time\_per\_record (in module *niscopes.Session*), 59  
HYSTERESIS (*niscopes.TriggerType* attribute), 137

## I

IMMEDIATE (*niscopes.TriggerType* attribute), 138  
import\_attribute\_configuration\_buffer() (in module *niscopes.Session*), 30  
import\_attribute\_configuration\_file() (in module *niscopes.Session*), 31  
IN\_PROGRESS (*niscopes.AcquisitionStatus* attribute), 126  
INCOMPLETE (*niscopes.RISMethod* attribute), 133  
INITIALIZE\_SERVER\_SESSION (*niscopes.SessionInitializationBehavior* attribute), 149  
initiate() (in module *niscopes.Session*), 31  
input\_clock\_source (in module *niscopes.Session*), 59  
input\_impedance (in module *niscopes.Session*), 60  
instrument\_firmware\_revision (in module *niscopes.Session*), 61  
instrument\_manufacturer (in module *niscopes.Session*), 61  
instrument\_model (in module *niscopes.Session*), 62  
instruments (*niscopes.Session.niscopes.Session* attribute), 126  
INTEGRAL (*niscopes.ClearableMeasurement* attribute), 130  
INTEGRAL (*niscopes.ScalarMeasurement* attribute), 136  
interleaving\_offset\_correction\_enabled (in module *niscopes.Session*), 62  
InvalidRepeatedCapabilityError, 142  
INVERSE (*niscopes.ArrayMeasurement* attribute), 127  
io\_resource\_descriptor (in module *niscopes.Session*), 63  
is\_probe\_comp\_on (in module *niscopes.Session*), 63

## L

LAST\_ACQ\_HISTOGRAM (*niscopes.ArrayMeasurement* attribute), 127  
LEAVING (*niscopes.TriggerWindowMode* attribute), 138  
LESS (*niscopes.GlitchCondition* attribute), 132  
LF\_REJECT (*niscopes.TriggerCoupling* attribute), 137

LIMITED\_BIN\_WIDTH (*niscopes.RISMethod* attribute), 134  
LINE\_NUMBER (*niscopes.VideoTriggerEvent* attribute), 140  
lock() (in module *niscopes.Session*), 31  
logical\_name (in module *niscopes.Session*), 64  
LOW\_REF\_VOLTS (*niscopes.ClearableMeasurement* attribute), 130  
LOW\_REF\_VOLTS (*niscopes.ScalarMeasurement* attribute), 136  
LOWHIGH (*niscopes.PercentageMethod* attribute), 133  
LOWPASS (*niscopes.FilterType* attribute), 132

## M

M\_PAL (*niscopes.VideoSignalFormat* attribute), 139  
master\_enable (in module *niscopes.Session*), 64  
max\_input\_frequency (in module *niscopes.Session*), 65  
max\_real\_time\_sampling\_rate (in module *niscopes.Session*), 65  
max\_ris\_rate (in module *niscopes.Session*), 66  
meas\_array\_gain (in module *niscopes.Session*), 66  
meas\_array\_offset (in module *niscopes.Session*), 67  
meas\_chan\_high\_ref\_level (in module *niscopes.Session*), 67  
meas\_chan\_low\_ref\_level (in module *niscopes.Session*), 68  
meas\_chan\_mid\_ref\_level (in module *niscopes.Session*), 69  
meas\_filter\_center\_freq (in module *niscopes.Session*), 69  
meas\_filter\_cutoff\_freq (in module *niscopes.Session*), 70  
meas\_filter\_order (in module *niscopes.Session*), 71  
meas\_filter\_ripple (in module *niscopes.Session*), 71  
meas\_filter\_taps (in module *niscopes.Session*), 72  
meas\_filter\_transient\_waveform\_percent (in module *niscopes.Session*), 72  
meas\_filter\_type (in module *niscopes.Session*), 73  
meas\_filter\_width (in module *niscopes.Session*), 74  
meas\_fir\_filter\_window (in module *niscopes.Session*), 74  
meas\_high\_ref (in module *niscopes.Session*), 75  
meas\_hysteresis\_percent (in module *niscopes.Session*), 75  
meas\_interpolation\_sampling\_factor (in module *niscopes.Session*), 76  
meas\_last\_acq\_histogram\_size (in module *niscopes.Session*), 76  
meas\_low\_ref (in module *niscopes.Session*), 77  
meas\_mid\_ref (in module *niscopes.Session*), 77  
meas\_other\_channel (in module *niscopes.Session*), 78  
meas\_percentage\_method (in module *niscopes.Session*), 78  
meas\_polynomial\_interpolation\_order (in module *niscopes.Session*), 79



- meas\_ref\_level\_units (in module *niscopesession*), 80  
 meas\_time\_histogram\_high\_time (in module *niscopesession*), 80  
 meas\_time\_histogram\_high\_volts (in module *niscopesession*), 81  
 meas\_time\_histogram\_low\_time (in module *niscopesession*), 82  
 meas\_time\_histogram\_low\_volts (in module *niscopesession*), 82  
 meas\_time\_histogram\_size (in module *niscopesession*), 83  
 meas\_voltage\_histogram\_high\_volts (in module *niscopesession*), 83  
 meas\_voltage\_histogram\_low\_volts (in module *niscopesession*), 84  
 meas\_voltage\_histogram\_size (in module *niscopesession*), 85  
 MID\_REF\_VOLTS (*niscopesession*.ClearableMeasurement attribute), 130  
 MID\_REF\_VOLTS (*niscopesession*.ScalarMeasurement attribute), 136  
 MIN\_NUM\_AVERAGES (*niscopesession*.RISMethod attribute), 133  
 min\_sample\_rate (in module *niscopesession*), 85  
 MINMAX (*niscopesession*.PercentageMethod attribute), 133  
 module  
     *niscopesession*, 8  
 MULTI\_ACQ\_AVERAGE (*niscopesession*.ArrayMeasurement attribute), 128  
 MULTI\_ACQ\_AVERAGE (*niscopesession*.ClearableMeasurement attribute), 129  
 MULTI\_ACQ\_TIME\_HISTOGRAM (*niscopesession*.ArrayMeasurement attribute), 127  
 MULTI\_ACQ\_TIME\_HISTOGRAM (*niscopesession*.ClearableMeasurement attribute), 129  
 MULTI\_ACQ\_VOLTAGE\_HISTOGRAM (*niscopesession*.ArrayMeasurement attribute), 127  
 MULTI\_ACQ\_VOLTAGE\_HISTOGRAM (*niscopesession*.ClearableMeasurement attribute), 129  
 MULTIPLY\_CHANNELS (*niscopesession*.ArrayMeasurement attribute), 127
- ## N
- NEGATIVE (*niscopesession*.GlitchPolarity attribute), 133  
 NEGATIVE (*niscopesession*.RuntPolarity attribute), 134  
 NEGATIVE (*niscopesession*.TriggerSlope attribute), 137  
 NEGATIVE (*niscopesession*.VideoPolarity attribute), 139  
 NEGATIVE (*niscopesession*.WidthPolarity attribute), 141  
*niscopesession*  
     module, 8  
 NO\_MEASUREMENT (*niscopesession*.ArrayMeasurement attribute), 127  
 NO\_MEASUREMENT (*niscopesession*.ScalarMeasurement attribute), 135  
 NO\_TRIGGER\_MOD (*niscopesession*.TriggerModifier attribute), 137  
 NONE (*niscopesession*.FIRFilterWindow attribute), 131  
 NONE (*niscopesession*.RuntimeCondition attribute), 135  
 NORMAL (*niscopesession*.AcquisitionType attribute), 127  
 NOW (*niscopesession*.FetchRelativeTo attribute), 131  
 NTSC (*niscopesession*.VideoSignalFormat attribute), 139
- ## O
- ON\_DEMAND (*niscopesession*.CableSenseMode attribute), 128  
 onboard\_memory\_size (in module *niscopesession*), 86  
 Option (class in *niscopesession*), 133  
 output\_clock\_source (in module *niscopesession*), 86  
 OUTSIDE (*niscopesession*.RuntimeCondition attribute), 135  
 OUTSIDE (*niscopesession*.WidthCondition attribute), 141  
 OVERSHOOT (*niscopesession*.ClearableMeasurement attribute), 129  
 OVERSHOOT (*niscopesession*.ScalarMeasurement attribute), 136
- ## P
- PAL (*niscopesession*.VideoSignalFormat attribute), 139  
 PERCENTAGE (*niscopesession*.RefLevelUnits attribute), 134  
 PercentageMethod (class in *niscopesession*), 133  
 PERIOD (*niscopesession*.ClearableMeasurement attribute), 129  
 PERIOD (*niscopesession*.ScalarMeasurement attribute), 135  
 PHASE\_DELAY (*niscopesession*.ClearableMeasurement attribute), 130  
 PHASE\_DELAY (*niscopesession*.ScalarMeasurement attribute), 136  
 pll\_lock\_status (in module *niscopesession*), 87  
 points\_done (in module *niscopesession*), 87  
 poll\_interval (in module *niscopesession*), 88  
 POLYNOMIAL\_INTERPOLATION (*niscopesession*.ArrayMeasurement attribute), 127  
 POSITIVE (*niscopesession*.GlitchPolarity attribute), 133  
 POSITIVE (*niscopesession*.RuntPolarity attribute), 134  
 POSITIVE (*niscopesession*.TriggerSlope attribute), 137  
 POSITIVE (*niscopesession*.VideoPolarity attribute), 139  
 POSITIVE (*niscopesession*.WidthPolarity attribute), 141  
 PRESHOOT (*niscopesession*.ClearableMeasurement attribute), 129  
 PRESHOOT (*niscopesession*.ScalarMeasurement attribute), 136  
 PRETRIGGER (*niscopesession*.FetchRelativeTo attribute), 131  
 probe\_attenuation (in module *niscopesession*), 88  
 probe\_compensation\_signal\_start() (in module *niscopesession*), 32  
 probe\_compensation\_signal\_stop() (in module *niscopesession*), 32
- ## R
- read() (in module *niscopesession*), 32  
 READ\_POINTER (*niscopesession*.FetchRelativeTo attribute), 131  
 ready\_for\_advance\_event\_output\_terminal (in module *niscopesession*), 89

[ready\\_for\\_advance\\_event\\_terminal\\_name](#) (in module *niscope.Session*), 89  
[ready\\_for\\_ref\\_event\\_output\\_terminal](#) (in module *niscope.Session*), 90  
[ready\\_for\\_ref\\_event\\_terminal\\_name](#) (in module *niscope.Session*), 90  
[ready\\_for\\_start\\_event\\_output\\_terminal](#) (in module *niscope.Session*), 91  
[ready\\_for\\_start\\_event\\_terminal\\_name](#) (in module *niscope.Session*), 91  
[record\\_arm\\_source](#) (in module *niscope.Session*), 92  
[records\\_done](#) (in module *niscope.Session*), 92  
[ref\\_clk\\_rate](#) (in module *niscope.Session*), 92  
[ref\\_trig\\_tdc\\_enable](#) (in module *niscope.Session*), 94  
[ref\\_trigger\\_detector\\_location](#) (in module *niscope.Session*), 93  
[ref\\_trigger\\_minimum\\_quiet\\_time](#) (in module *niscope.Session*), 93  
[ref\\_trigger\\_terminal\\_name](#) (in module *niscope.Session*), 94  
[REFERENCE](#) (*niscope.WhichTrigger* attribute), 140  
[RefLevelUnits](#) (class in *niscope*), 134  
[RefTriggerDetectorLocation](#) (class in *niscope*), 134  
[reset\(\)](#) (in module *niscope.Session*), 34  
[reset\\_device\(\)](#) (in module *niscope.Session*), 34  
[reset\\_with\\_defaults\(\)](#) (in module *niscope.Session*), 34  
[resolution](#) (in module *niscope.Session*), 95  
[RESTORE\\_EXTERNAL\\_CALIBRATION](#) (*niscope.Option* attribute), 133  
[ris\\_in\\_auto\\_setup\\_enable](#) (in module *niscope.Session*), 95  
[ris\\_method](#) (in module *niscope.Session*), 96  
[ris\\_num\\_averages](#) (in module *niscope.Session*), 96  
[RISE\\_SLEW\\_RATE](#) (*niscope.ClearableMeasurement* attribute), 129  
[RISE\\_SLEW\\_RATE](#) (*niscope.ScalarMeasurement* attribute), 136  
[RISE\\_TIME](#) (*niscope.ClearableMeasurement* attribute), 129  
[RISE\\_TIME](#) (*niscope.ScalarMeasurement* attribute), 135  
[RISMethod](#) (class in *niscope*), 133  
[RpcError](#), 142  
[RUNT](#) (*niscope.TriggerType* attribute), 138  
[runt\\_high\\_threshold](#) (in module *niscope.Session*), 97  
[runt\\_low\\_threshold](#) (in module *niscope.Session*), 97  
[runt\\_polarity](#) (in module *niscope.Session*), 98  
[runt\\_time\\_condition](#) (in module *niscope.Session*), 98  
[runt\\_time\\_high\\_limit](#) (in module *niscope.Session*), 99  
[runt\\_time\\_low\\_limit](#) (in module *niscope.Session*), 99  
[RuntPolarity](#) (class in *niscope*), 134  
[RuntimeCondition](#) (class in *niscope*), 135

## S

[samp\\_clk\\_timebase\\_div](#) (in module *niscope.Session*), 100  
[samp\\_clk\\_timebase\\_rate](#) (in module *niscope.Session*), 101  
[samp\\_clk\\_timebase\\_src](#) (in module *niscope.Session*), 101  
[sample\\_clock\\_timebase\\_multiplier](#) (in module *niscope.Session*), 100  
[sample\\_mode](#) (in module *niscope.Session*), 100  
[ScalarMeasurement](#) (class in *niscope*), 135  
[SECAM](#) (*niscope.VideoSignalFormat* attribute), 139  
[self\\_cal\(\)](#) (in module *niscope.Session*), 34  
[SELF\\_CALIBRATE\\_ALL\\_CHANNELS](#) (*niscope.Option* attribute), 133  
[self\\_test\(\)](#) (in module *niscope.Session*), 35  
[SelfTestError](#), 142  
[send\\_software\\_trigger\\_edge\(\)](#) (in module *niscope.Session*), 35  
[serial\\_number](#) (in module *niscope.Session*), 102  
[Session](#) (class in *niscope*), 8  
[SessionInitializationBehavior](#) (class in *niscope*), 149  
[simulate](#) (in module *niscope.Session*), 104  
[SINGLE\\_ENDED](#) (*niscope.TerminalConfiguration* attribute), 136  
[SIXTEEN\\_TAP\\_HANNING](#) (*niscope.FlexFIRAntialiasFilterType* attribute), 132  
[SLOPE\\_EITHER](#) (*niscope.TriggerSlope* attribute), 137  
[SOFTWARE](#) (*niscope.TriggerType* attribute), 138  
[specific\\_driver\\_description](#) (in module *niscope.Session*), 104  
[specific\\_driver\\_revision](#) (in module *niscope.Session*), 105  
[specific\\_driver\\_vendor](#) (in module *niscope.Session*), 105  
[START](#) (*niscope.FetchRelativeTo* attribute), 131  
[START](#) (*niscope.WhichTrigger* attribute), 140  
[start\\_to\\_ref\\_trigger\\_holdoff](#) (in module *niscope.Session*), 105  
[start\\_trigger\\_terminal\\_name](#) (in module *niscope.Session*), 106  
[STATUS\\_UNKNOWN](#) (*niscope.AcquisitionStatus* attribute), 126  
[SUBTRACT\\_CHANNELS](#) (*niscope.ArrayMeasurement* attribute), 128  
[supported\\_instrument\\_models](#) (in module *niscope.Session*), 106

## T

[tclk](#) (in module *niscope.Session*), 119  
[TerminalConfiguration](#) (class in *niscope*), 136

- TIME\_DELAY (*niscopes.ClearableMeasurement* attribute), 130
- TIME\_DELAY (*niscopes.ScalarMeasurement* attribute), 136
- TIME\_HISTOGRAM\_HITS (*niscopes.ClearableMeasurement* attribute), 131
- TIME\_HISTOGRAM\_MAX (*niscopes.ClearableMeasurement* attribute), 130
- TIME\_HISTOGRAM\_MEAN (*niscopes.ClearableMeasurement* attribute), 130
- TIME\_HISTOGRAM\_MEAN\_PLUS\_2\_STDEV (*niscopes.ClearableMeasurement* attribute), 131
- TIME\_HISTOGRAM\_MEAN\_PLUS\_3\_STDEV (*niscopes.ClearableMeasurement* attribute), 131
- TIME\_HISTOGRAM\_MEAN\_PLUS\_STDEV (*niscopes.ClearableMeasurement* attribute), 131
- TIME\_HISTOGRAM\_MEDIAN (*niscopes.ClearableMeasurement* attribute), 130
- TIME\_HISTOGRAM\_MIN (*niscopes.ClearableMeasurement* attribute), 131
- TIME\_HISTOGRAM\_MODE (*niscopes.ClearableMeasurement* attribute), 130
- TIME\_HISTOGRAM\_NEW\_HITS (*niscopes.ClearableMeasurement* attribute), 131
- TIME\_HISTOGRAM\_PEAK\_TO\_PEAK (*niscopes.ClearableMeasurement* attribute), 131
- TIME\_HISTOGRAM\_STDEV (*niscopes.ClearableMeasurement* attribute), 130
- TRIANGLE (*niscopes.FIRFilterWindow* attribute), 131
- TRIANGLE\_WINDOW (*niscopes.ArrayMeasurement* attribute), 128
- TRIGGER (*niscopes.FetchRelativeTo* attribute), 132
- trigger\_auto\_triggered (in module *niscopes.Session*), 107
- trigger\_coupling (in module *niscopes.Session*), 107
- trigger\_delay\_time (in module *niscopes.Session*), 108
- trigger\_holdoff (in module *niscopes.Session*), 108
- trigger\_hysteresis (in module *niscopes.Session*), 109
- trigger\_impedance (in module *niscopes.Session*), 109
- trigger\_level (in module *niscopes.Session*), 110
- trigger\_modifier (in module *niscopes.Session*), 110
- trigger\_slope (in module *niscopes.Session*), 111
- trigger\_source (in module *niscopes.Session*), 111
- trigger\_type (in module *niscopes.Session*), 111
- trigger\_window\_high\_level (in module *niscopes.Session*), 112
- trigger\_window\_low\_level (in module *niscopes.Session*), 113
- trigger\_window\_mode (in module *niscopes.Session*), 113
- TriggerCoupling (class in *niscopes*), 137
- TriggerModifier (class in *niscopes*), 137
- TriggerSlope (class in *niscopes*), 137
- TriggerType (class in *niscopes*), 137
- TriggerWindowMode (class in *niscopes*), 138
- TV (*niscopes.TriggerType* attribute), 138
- tv\_trigger\_event (in module *niscopes.Session*), 114
- tv\_trigger\_line\_number (in module *niscopes.Session*), 114
- tv\_trigger\_polarity (in module *niscopes.Session*), 115
- tv\_trigger\_signal\_format (in module *niscopes.Session*), 115
- ## U
- UNBALANCED\_DIFFERENTIAL (*niscopes.TerminalConfiguration* attribute), 136
- unlock() (in module *niscopes.Session*), 36
- UnsupportedConfigurationError, 141
- use\_spec\_initial\_x (in module *niscopes.Session*), 115
- ## V
- vertical\_coupling (in module *niscopes.Session*), 116
- vertical\_offset (in module *niscopes.Session*), 116
- vertical\_range (in module *niscopes.Session*), 117
- VerticalCoupling (class in *niscopes*), 139
- VIDEO\_1080I\_50\_FIELDS\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 140
- VIDEO\_1080I\_59\_94\_FIELDS\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 140
- VIDEO\_1080I\_60\_FIELDS\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 140
- VIDEO\_1080P\_24\_FRAMES\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 140
- VIDEO\_480I\_59\_94\_FIELDS\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 139
- VIDEO\_480I\_60\_FIELDS\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 139
- VIDEO\_480P\_59\_94\_FRAMES\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 139
- VIDEO\_480P\_60\_FRAMES\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 139
- VIDEO\_576I\_50\_FIELDS\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 139
- VIDEO\_576P\_50\_FRAMES\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 139
- VIDEO\_720P\_50\_FRAMES\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 140
- VIDEO\_720P\_59\_94\_FRAMES\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 140
- VIDEO\_720P\_60\_FRAMES\_PER\_SECOND (*niscopes.VideoSignalFormat* attribute), 140
- VideoPolarity (class in *niscopes*), 139
- VideoSignalFormat (class in *niscopes*), 139
- VideoTriggerEvent (class in *niscopes*), 140
- VOLTAGE\_AVERAGE (*niscopes.ClearableMeasurement* attribute), 129
- VOLTAGE\_AVERAGE (*niscopes.ScalarMeasurement* attribute), 135

VOLTAGE\_BASE (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_BASE (*niscopes.ScalarMeasurement* attribute), 136  
VOLTAGE\_BASE\_TO\_TOP (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_BASE\_TO\_TOP (*niscopes.ScalarMeasurement* attribute), 136  
VOLTAGE\_CYCLE\_AVERAGE (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_CYCLE\_AVERAGE (*niscopes.ScalarMeasurement* attribute), 135  
VOLTAGE\_CYCLE\_RMS (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_CYCLE\_RMS (*niscopes.ScalarMeasurement* attribute), 135  
VOLTAGE\_HIGH (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_HIGH (*niscopes.ScalarMeasurement* attribute), 135  
VOLTAGE\_HISTOGRAM\_HITS (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_MAX (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_MEAN (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_MEAN\_PLUS\_2\_STDEV (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_MEAN\_PLUS\_3\_STDEV (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_MEAN\_PLUS\_STDEV (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_MEDIAN (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_MIN (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_MODE (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_NEW\_HITS (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_PEAK\_TO\_PEAK (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_HISTOGRAM\_STDEV (*niscopes.ClearableMeasurement* attribute), 130  
VOLTAGE\_LOW (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_LOW (*niscopes.ScalarMeasurement* attribute), 135  
VOLTAGE\_MAX (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_MAX (*niscopes.ScalarMeasurement* attribute), 135  
VOLTAGE\_MIN (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_MIN (*niscopes.ScalarMeasurement* attribute), 135  
VOLTAGE\_PEAK\_TO\_PEAK (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_PEAK\_TO\_PEAK (*niscopes.ScalarMeasurement* attribute), 135  
VOLTAGE\_RMS (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_RMS (*niscopes.ScalarMeasurement* attribute), 135  
VOLTAGE\_TOP (*niscopes.ClearableMeasurement* attribute), 129  
VOLTAGE\_TOP (*niscopes.ScalarMeasurement* attribute), 136  
VOLTS (*niscopes.RefLevelUnits* attribute), 134

## W

WhichTrigger (class in *niscopes*), 140  
WIDTH (*niscopes.TriggerType* attribute), 138  
width\_condition (in module *niscopes.Session*), 118  
width\_high\_threshold (in module *niscopes.Session*), 118  
width\_low\_threshold (in module *niscopes.Session*), 119  
WIDTH\_NEG (*niscopes.ClearableMeasurement* attribute), 130  
WIDTH\_NEG (*niscopes.ScalarMeasurement* attribute), 135  
width\_polarity (in module *niscopes.Session*), 119  
WIDTH\_POS (*niscopes.ClearableMeasurement* attribute), 130  
WIDTH\_POS (*niscopes.ScalarMeasurement* attribute), 135  
WidthCondition (class in *niscopes*), 141  
WidthPolarity (class in *niscopes*), 141  
WINDOW (*niscopes.TriggerType* attribute), 138  
WINDOWED\_FIR\_FILTER (*niscopes.ArrayMeasurement* attribute), 128  
WITHIN (*niscopes.RunTimeCondition* attribute), 135  
WITHIN (*niscopes.WidthCondition* attribute), 141