NI-SCOPE Python API Documentation

Release 1.4.7

NI

Dec 15, 2023

DOCUMENTATION

CHAPTER

ONE

ABOUT

The **niscope** module provides a Python API for NI-SCOPE. The code is maintained in the Open Source repository for nimi-python.

1.1 Support Policy

niscope supports all the Operating Systems supported by NI-SCOPE.

It follows Python Software Foundation support policy for different versions of CPython.

CHAPTER TWO

CONTRIBUTING

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

CHAPTER

THREE

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.

CHAPTER

FOUR

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 niscope module

4.1.1 Installation

As a prerequisite to using the **niscope** 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 niscope~=1.4.7
```

4.1.2 Usage

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

```
import niscope
with niscope.Session("Dev1") as session:
    session.channels[0].configure_vertical(range=1.0, coupling=niscope.VerticalCoupling.
↪AC)
    session.channels[1].configure_vertical(range=10.0, coupling=niscope.VerticalCoupling.
↪DC)
    session.configure_horizontal_timing(min_sample_rate=50000000, 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 {0}, record {1} samples acquired: {2:,}\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 niscope.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 $V_{pk\text{-}pk}$ 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.ClearableMeasureme*

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.ClearableMeasurement*) – 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 `niscope.Session.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

niscope.Session.**configure_trigger_digital**(*trigger_source*, *slope=niscope.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

`niscope.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 *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.
- **signal_format** (*niscope.VideoSignalFormat*) – Specifies the type of video signal sync the digitizer should look for. Refer to *niscope.Session.tv_trigger_signal_format* for more information.
- **event** (*niscope.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** (*niscope.VideoPolarity*) – Specifies the polarity of the video signal sync.
- **trigger_coupling** (*niscope.TriggerCoupling*) – Applies coupling and filtering options to the trigger signal. Refer to *niscope.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 *niscope.Session.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 *niscope.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 *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_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

niscope.Session.**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 `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_vertical()`

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

Example: `my_session.configure_vertical()`

Parameters

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

disable

niscope.Session.**disable**()

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

export_attribute_configuration_buffer

niscope.Session.**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** (*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

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:

voltage = binary data × gain factor + offset

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

voltage = binary data × gain factor + 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.in16*

– *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.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_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_RTSI_0` ('VAL_RTSI_0') - RTSI 0 `NISCOPE_VAL_RTSI_1` ('VAL_RTSI_1') - RTSI 1 `NISCOPE_VAL_RTSI_2` ('VAL_RTSI_2') - RTSI 2 `NISCOPE_VAL_RTSI_3` ('VAL_RTSI_3') - RTSI 3 `NISCOPE_VAL_RTSI_4` ('VAL_RTSI_4') - RTSI 4 `NISCOPE_VAL_RTSI_5` ('VAL_RTSI_5') - RTSI 5 `NISCOPE_VAL_RTSI_6` ('VAL_RTSI_6') - RTSI 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 specificed 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 that 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	enums.CableSenseMode
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	enums.TerminalConfiguration
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	int
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	enums.FlexFIRAntialiasFilterType
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	str
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	enums.GlitchCondition
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	enums.GlitchPolarity
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	float
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	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: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	enums.FilterType
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* ± 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	enums.PercentageMethod
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	int
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	enums.RefLevelUnits
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: -5.0e-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_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 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	enums.RefTriggerDetectorLocation
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	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: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 autosetup. Valid Values: True (1) - Use RIS sample rates in autosetup False (0) - Do not use RIS sample rates in autosetup

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	enums.RISMethod
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 oversampling 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	int
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**

runt_high_threshold

niscope.Session.**runt_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.runt_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**

runt_low_threshold

niscope.Session.**runt_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.runt_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**

runt_polarity

niscope.Session.**runt_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.runt_low_threshold* in a positive direction;
- The trailing edge of the pulse crosses the *niscope.Session.runt_low_threshold* in a negative direction; and
- No portion of the pulse crosses the *niscope.Session.runt_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.runt_high_threshold* in a negative direction;
- The trailing edge of the pulse crosses the *niscope.Session.runt_high_threshold* in a positive direction; and
- No portion of the pulse crosses the *niscope.Session.runt_low_threshold*.

When set to *EITHER*, the oscilloscope triggers in either case.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NISCOPE_ATTR_RUNT_POLARITY**

runt_time_condition

niscope.Session.**runt_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.runt_time_low_limit* and *niscope.Session.runt_time_high_limit* properties.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.RuntTimeCondition
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	enums.TriggerCoupling
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	enums.TriggerModifier
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	enums.TriggerSlope
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	str
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 <= Vertical Range/2 + Vertical Offset High Trigger Level >= (-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 <= Vertical Range/2 + Vertical Offset Low Trigger Level >= (-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	enums.TriggerWindowMode
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	enums.VideoTriggerEvent
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	int
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	enums.VideoPolarity
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	enums.VideoSignalFormat
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	bool
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	enums.VerticalCoupling
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	enums.WidthCondition
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	float
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	enums.WidthPolarity
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

- *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*

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 niscope.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 niscope.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 niscope.**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 niscope.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 niscope.**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 niscope.**FetchRelativeTo**

READ_POINTER

The read pointer is set to zero when a new acquisition is initiated. After every fetch the read pointer is incremeted 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 *niscope.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 niscope.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 niscope.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 niscope.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 niscope.**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 niscope.**Option**

SELF_CALIBRATE_ALL_CHANNELS
Self Calibrating all Channels

RESTORE_EXTERNAL_CALIBRATION
Restore External Calibration.

PercentageMethod

class niscope.**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 niscope.**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 niscope.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 niscope.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 niscope.RuntPolarity`

`POSITIVE`

Trigger on pulses of positive polarity relative to *`niscope.Session.runt_low_threshold`* that do not cross *`niscope.Session.runt_high_threshold`*.

`NEGATIVE`

Trigger on pulses of negative polarity relative to *`niscope.Session.runt_high_threshold`* that do not cross *`niscope.Session.runt_low_threshold`*.

`EITHER`

Trigger on pulses of either positive or negative polarity.

RuntTimeCondition

class niscope.**RuntTimeCondition**

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 *niscope.Session.runt_time_low_limit* and *niscope.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 *niscope.Session.runt_time_low_limit* and *niscope.Session.runt_time_high_limit*.

ScalarMeasurement

class niscope.**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 niscope.TerminalConfiguration

SINGLE_ENDED

Channel is single ended

UNBALANCED_DIFFERENTIAL

Channel is unbalanced differential

DIFFERENTIAL

Channel is differential

TriggerCoupling

class niscope.**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 niscope.**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 niscope.**TriggerSlope**

NEGATIVE

Falling edge

POSITIVE

Rising edge

SLOPE_EITHER

Either edge

TriggerType

class niscope.**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 `niscope.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 niscope.**VerticalCoupling**

AC
AC coupling

DC
DC coupling

GND
GND coupling

VideoPolarity

class niscope.**VideoPolarity**

POSITIVE
Specifies that the video signal has positive polarity.

NEGATIVE
Specifies that the video signal has negative polarity.

VideoSignalFormat

class niscope.**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 niscope.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 niscope.WhichTrigger`

START

ARM_REFERENCE

REFERENCE

ADVANCE

WidthCondition

class niscope.**WidthCondition**

WITHIN

Trigger on pulses with a duration within the range bounded by *niscope.Session.width_low_threshold* and *niscope.Session.width_high_threshold*.

OUTSIDE

Trigger on pulses with a duration not within the range bounded by *niscope.Session.width_low_threshold* and *niscope.Session.width_high_threshold*.

WidthPolarity

class niscope.**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 niscope.errors.**Error**

Base exception type that all NI-SCOPE exceptions derive from

DriverError

exception niscope.errors.**DriverError**

An error originating from the NI-SCOPE driver

UnsupportedConfigurationError

exception niscope.errors.**UnsupportedConfigurationError**

An error due to using this module in an usupported 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)

```python
#!/usr/bin/python

import argparse
import niscope
import pprint
import sys

pp = pprint.PrettyPrinter(indent=4, width=80)


def example(resource_name, channels, options, length, voltage):
    with niscope.Session(resource_name=resource_name, options=options) as session:
        session.configure_vertical(range=voltage, coupling=niscope.VerticalCoupling.AC)
        session.configure_horizontal_timing(min_sample_rate=50000000, min_num_pts=length, ref_position=50.0, num_records=1, enforce_realtime=True)
        with session.initiate():
            waveforms = session.channels[channels].fetch(num_samples=length)
        for i in range(len(waveforms)):
            print('Waveform {0} information:'.format(i))
            print(str(waveforms[i]) + '\n\n')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Acquires one record from the given channels.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource name of an NI digitizer.')
    parser.add_argument('-c', '--channels', default='0', help='Channel(s) to use')
    parser.add_argument('-l', '--length', default=1000, type=int, help='Measure record length')
    parser.add_argument('-v', '--voltage', default=1.0, type=float, help='Voltage range (V)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.channels, args.option_string, args.length, args.voltage)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe', }, }
    example('PXI1Slot2', '0', options, 1000, 1.0)
```

(continues on next page)

(continued from previous page)

```
def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe',
↪]
    _main(cmd_line)


if __name__ == '__main__':
    main()
```

niscope_fetch_forever.py

Listing 2: (niscope_fetch_forever.py)

```
#!/usr/bin/python

import argparse
import hightime
import niscope
import numpy as np
import pprint
import sys


pp = pprint.PrettyPrinter(indent=4, width=80)


# We use fetch_into which allows us to allocate a single buffer per channel and "fetch
↪into" it a section at a time without having to
# reconstruct the waveform once we are done
def example(resource_name, options, total_acquisition_time_in_seconds, voltage, sample_
↪rate_in_hz, samples_per_fetch):
    total_samples = int(total_acquisition_time_in_seconds * sample_rate_in_hz)
    # 1. Opening session
    with niscope.Session(resource_name=resource_name, options=options) as session:
        # We will acquire on all channels of the device
        channel_list = [c for c in range(session.channel_count)]  # Need an actual list
↪and not a range

        # 2. Creating numpy arrays
        waveforms = [np.ndarray(total_samples, dtype=np.float64) for c in channel_list]

        # 3. Configuring
        session.configure_horizontal_timing(min_sample_rate=sample_rate_in_hz, min_num_
↪pts=1, ref_position=0.0, num_records=1, enforce_realtime=True)
        session.channels[channel_list].configure_vertical(voltage, coupling=niscope.
↪VerticalCoupling.DC, enabled=True)
        # Configure software trigger, but never send the trigger.
        # This starts an infinite acquisition, until you call session.abort() or session.
↪close()
```

(continues on next page)

(continued from previous page)

```python
        session.configure_trigger_software()
        current_pos = 0
        # 4. initiating
        with session.initiate():
            while current_pos < total_samples:
                # We fetch each channel at a time so we don't have to de-interleave afterwards
                # We do not keep the wfm_info returned from fetch_into
                for channel, waveform in zip(channel_list, waveforms):
                    # 5. fetching - we return the slice of the waveform array that we want to "fetch into"
                    session.channels[channel].fetch_into(waveform[current_pos:current_pos + samples_per_fetch], relative_to=niscope.FetchRelativeTo.READ_POINTER,
                                                         offset=0, record_number=0, num_records=1, timeout=hightime.timedelta(seconds=5.0))
                current_pos += samples_per_fetch


def _main(argsv):
    parser = argparse.ArgumentParser(description='Fetch more samples than will fit in memory.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource name of an NI digitizer.')
    parser.add_argument('-t', '--time', default=10, type=int, help='Time to sample (s)')
    parser.add_argument('-v', '--voltage', default=1.0, type=float, help='Voltage range (V)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option string')
    parser.add_argument('-r', '--sample-rate', default=1000.0, type=float, help='Sample Rate (Hz)')
    parser.add_argument('-s', '--samples-per-fetch', default=100, type=int, help='Samples per fetch')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.time, args.voltage, args.sample_rate, args.samples_per_fetch)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe', }, }
    example('PXI1Slot2', options, 10, 1.0, 1000.0, 100)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe', ]
    _main(cmd_line)

```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    main()
```

niscope_fetch_into.py

Listing 3: (niscope_fetch_into.py)

```
#!/usr/bin/python

import argparse
import niscope
import numpy
import pprint
import sys

pp = pprint.PrettyPrinter(indent=4, width=80)


def example(resource_name, channels, options, length, voltage):
    # fetch_into() allows you to preallocate and reuse the destination of the fetched␣
↪waveforms, which can result in better performance at the expense of the usability of␣
↪fetch().
    channels = [ch.strip() for ch in channels.split(",")]
    num_channels = len(channels)
    num_records = 5
    total_num_wfms = num_channels * num_records
    # preallocate a single array for all samples in all waveforms
    # Supported array types are: numpy.float64, numpy.int8, numpy.int16, numpy.int32
    # int8, int16, int32 are for fetching unscaled data, which is the fastest way to␣
↪fetch.
    # Gain and Offset are stored in the returned WaveformInfo objects and can be applied␣
↪to the data by the user later.
    wfm = numpy.ndarray(length * total_num_wfms, dtype=numpy.float64)
    with niscope.Session(resource_name=resource_name, options=options) as session:
        session.configure_vertical(range=voltage, coupling=niscope.VerticalCoupling.AC)
        session.configure_horizontal_timing(min_sample_rate=50000000, min_num_pts=length,
↪ ref_position=50.0, num_records=num_records, enforce_realtime=True)
        with session.initiate():
            waveforms = session.channels[channels].fetch_into(waveform=wfm, num_
↪records=num_records)
        for i in range(len(waveforms)):
            print(f'Waveform {i} information:')
            print(f'{waveforms[i]}\n\n')
            print(f'Samples: {waveforms[i].samples.tolist()}')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Fetches data directly into a␣
```

(continues on next page)

(continued from previous page)

```
↪preallocated numpy array.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of an NI digitizer.')
    parser.add_argument('-c', '--channels', default='0', help='Channel(s) to use')
    parser.add_argument('-l', '--length', default=100, type=int, help='Measure record␣
↪length')
    parser.add_argument('-v', '--voltage', default=1.0, type=float, help='Voltage range␣
↪(V)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.channels, args.option_string, args.length, args.
↪voltage)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe', }
↪, }
    example('PXI1Slot2', '0, 1', options, 100, 1.0)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe',␣
↪]
    _main(cmd_line)


if __name__ == '__main__':
    main()

```

niscope_read.py

Listing 4: (niscope_read.py)

```
#!/usr/bin/python

import argparse
import niscope
import pprint
import sys

pp = pprint.PrettyPrinter(indent=4, width=80)


def example(resource_name, channels, options, length, voltage):
```

(continues on next page)

(continued from previous page)

```python
    with niscope.Session(resource_name=resource_name, options=options) as session:
        session.configure_vertical(range=voltage, coupling=niscope.VerticalCoupling.AC)
        session.configure_horizontal_timing(min_sample_rate=50000000, min_num_pts=length, ref_position=50.0, num_records=1, enforce_realtime=True)
        waveforms = session.channels[channels].read(num_samples=length)
        for i in range(len(waveforms)):
            print('Waveform {0} information:'.format(i))
            print(str(waveforms[i]) + '\n\n')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Acquires one record from the given channels.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource name of an NI digitizer.')
    parser.add_argument('-c', '--channels', default='0', help='Channel(s) to use')
    parser.add_argument('-l', '--length', default=1000, type=int, help='Measure record length')
    parser.add_argument('-v', '--voltage', default=1.0, type=float, help='Voltage range (V)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.channels, args.option_string, args.length, args.voltage)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe', }, }
    example('PXI1Slot2', '0', options, 1000, 1.0)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe', ]
    _main(cmd_line)


if __name__ == '__main__':
    main()
```

gRPC Support

Support for using NI-SCOPE over gRPC

SessionInitializationBehavior

`class niscope.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 niscope.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** (*niscope.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.

CHAPTER

FIVE

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).

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

A

B

C

D

E

F

G

H

I

L

M

S

T

U

V

W