NI Modular Instruments Python API Documentation

Release 1.4.0

National Instruments

Jul 09, 2021

Drivers

CHAPTER 1

About

The **nimi-python** repository generates Python bindings (Application Programming Interface) for interacting with the Modular Instrument drivers. The following drivers are supported:

- NI-DCPower (Python module: nidcpower)
- NI-Digital Pattern Driver (Python module: nidigital)
- NI-DMM (Python module: nidmm)
- NI-FGEN (Python module: nifgen)
- NI-ModInst (Python module: nimodinst)
- NI-SCOPE (Python module: niscope)
- NI Switch Executive (Python module: nise)
- NI-SWITCH (Python module: niswitch)
- NI-TClk (Python module: nitclk)

It is implemented as a set of Mako templates and per-driver metafiles that produce a Python module for each driver. The driver is called through its public C API using the ctypes Python library.

nimi-python supports all the Operating Systems supported by the underlying driver.

nimi-python follows Python Software Foundation support policy for different versions. At this time this includes Python 3.5 and above using CPython.

CHAPTER 2

Installation

Driver specific installation instructions can be found on Read The Docs:

- nidcpower
- nidigital
- nidmm
- nifgen
- nimodinst
- niscope
- nise
- niswitch
- nitclk

CHAPTER 3

Contributing

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

CHAPTER 4

Support / Feedback

The packages included in **nimi-python** package are supported by NI. For support, open a request through the NI support portal at ni.com.

CHAPTER 5

Bugs / Feature Requests

To report a bug or submit a feature request specific to NI Modular Instruments Python bindings (nimi-python), please use the GitHub issues page.

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

For hardware support or any other questions not specific to this GitHub project, please visit NI Community Forums.

CHAPTER 6

Documentation

Documentation is available here.

6.1 Additional Documentation

Refer to your driver documentation for device-specific information and detailed API documentation.

CHAPTER 7

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.

7.1 nidcpower module

7.1.1 Installation

As a prerequisite to using the nidcpower module, you must install the NI-DCPower runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

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

```
$ python -m pip install nidcpower~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install nidcpower
```

7.1.2 Usage

The following is a basic example of using the **nidcpower** module to open a session to a Source Meter Unit and measure voltage and current.

```python
import nidcpower
# Configure the session.

with nidcpower.Session(resource_name='PXI1Slot2/0') as session:
    session.measure_record_length = 20
    session.measure_record_length_is_finite = True
    session.measure_when = nidcpower.MeasureWhen.AUTOMATICALLY_AFTER_SOURCE_COMPLETE
```

(continues on next page)

(continued from previous page)

```
    session.voltage_level = 5.0

    session.commit()
    print('Effective measurement rate: {0} S/s'.format(session.measure_record_delta_
↪time / 1))

    samples_acquired = 0
    print('Channel           Num  Voltage    Current    In Compliance')
    row_format = '{0:15} {1:3d}    {2:8.6f}   {3:8.6f}   {4}'
    with session.initiate():
        channel_indices = '0-{0}'.format(session.channel_count - 1)
        channels = session.get_channel_names(channel_indices)
        for i, channel_name in enumerate(channels):
            samples_acquired = 0
            while samples_acquired < 20:
                measurements = session.channels[channel_name].fetch_
↪multiple(count=session.fetch_backlog)
                samples_acquired += len(measurements)
                for i in range(len(measurements)):
                    print(row_format.format(channel_name, i, measurements[i].voltage,␣
↪measurements[i].current, measurements[i].in_compliance))
```

Additional examples for NI-DCPower are located in src/nidcpower/examples/ directory.

7.1.3 API Reference

Session

class nidcpower.**Session**(*self*, *resource_name*, *channels=None*, *reset=False*, *options={}*, *independent_channels=True*)

Creates and returns a new NI-DCPower session to the instrument(s) and channel(s) specified in **resource name** to be used in all subsequent NI-DCPower method calls. With this method, you can optionally set the initial state of the following session properties:

- *nidcpower.Session.simulate*
- *nidcpower.Session.driver_setup*

After calling this method, the specified channel or channels will be in the Uncommitted state.

To place channel(s) in a known start-up state when creating a new session, set **reset** to True. This action is equivalent to using the *nidcpower.Session.reset()* method immediately after initializing the session.

To open a session and leave the channel(s) in an existing configuration without passing through a transitional output state, set **reset** to False. Next, configure the channel(s) as in the previous session, change the desired settings, and then call the *nidcpower.Session.initiate()* method to write both settings.

Details of Independent Channel Operation

With this method and channel-based NI-DCPower methods and properties, you can use any channels in the session independently. For example, you can initiate a subset of channels in the session with *nidcpower.Session.initiate()*, and the other channels in the session remain in the Uncommitted state.

When you initialize with independent channels, each channel steps through the NI-DCPower programming state model independently of all other channels, and you can specify a subset of channels for most operations.

Note You can make concurrent calls to a session from multiple threads, but the session executes the calls one at a time. If you specify multiple channels for a method or property, the session may perform the operation on multiple channels in parallel, though this is not guaranteed, and some operations may execute sequentially.

Parameters

- **`resource_name`** (*`str`*, *`list`*, *`tuple`*) – Specifies the **resource name** as seen in Measurement & Automation Explorer (MAX) or lsni, for example "PXI1Slot3" where "PXI1Slot3" is an instrument's **resource name**. If independent_channels is False, **resource name** can also be a logical IVI name.

  If independent_channels is True, **resource name** can be names of the instrument(s) and the channel(s) to initialize. Specify the instrument(s) and channel(s) using the form "PXI1Slot3/0,PXI1Slot3/2-3,PXI1Slot4/2-3 or PXI1Slot3/0,PXI1Slot3/2:3,PXI1Slot4/2:3", where "PXI1Slot3" and "PXI1Slot4" are instrument resource names followed by channels. If you exclude a channels string after an instrument resource name, all channels of the instrument(s) are included in the session.

- **`channels`** (*`str`*, *`list`*, *`range`*, *`tuple`*) – For new applications, use the default value of None and specify the channels in **resource name**.

  Specifies which output channel(s) to include in a new session. Specify multiple channels by using a channel list or a channel range. A channel list is a comma (,) separated sequence of channel names (for example, 0,2 specifies channels 0 and 2). A channel range is a lower bound channel followed by a hyphen (-) or colon (:) followed by an upper bound channel (for example, 0-2 specifies channels 0, 1, and 2).

  If independent_channels is False, this argument specifies which channels to include in a legacy synchronized channels session. If you do not specify any channels, by default all channels on the device are included in the session.

  If independent_channels is True, this argument combines with **resource name** to specify which channels to include in an independent channels session. Initializing an independent channels session with a channels argument is deprecated.

- **`reset`** (*`bool`*) – Specifies whether to reset channel(s) during the initialization procedure.

- **`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	{}

- **`independent_channels`** (*`bool`*) – Specifies whether to initialize the session with independent channels. Set this argument to False on legacy applications or if you are unable to upgrade your NI-DCPower driver runtime to 20.6 or higher.

Methods

abort

`nidcpower.Session.abort()`

Transitions the specified channel(s) from the Running state to the Uncommitted state. If a sequence is running, it is stopped. Any configuration methods called after this method are not applied until the `nidcpower.Session.initiate()` method is called. If power output is enabled when you call the `nidcpower.Session.abort()` method, the output channels remain in their current state and continue providing power.

Use the `nidcpower.Session.ConfigureOutputEnabled()` method to disable power output on a per channel basis. Use the `nidcpower.Session.reset()` method to disable output on all channels.

Refer to the Programming States topic in the *NI DC Power Supplies and SMUs Help* for information about the specific NI-DCPower software states.

Related Topics:

Programming States

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

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].abort()`

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

Example: `my_session.abort()`

clear_latched_output_cutoff_state

`nidcpower.Session.clear_latched_output_cutoff_state(output_cutoff_reason)`

Clears the state of an output cutoff that was engaged. To clear the state for all output cutoff reasons, use `ALL`.

Tip: This method can be called on specific channels within your `nidcpower.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_latched_output_cutoff_state()`

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

Example: `my_session.clear_latched_output_cutoff_state()`

Parameters **`output_cutoff_reason`** (`nidcpower.OutputCutoffReason`) – Specifies the reasons for which to clear the output cutoff state.

ALL	Clears all output cutoff conditions
VOLTAGE_OUTPUT_HIGH	Clears cutoffs caused when the output exceeded the high cutoff limit for voltage output
VOLTAGE_OUTPUT_LOW	Clears cutoffs caused when the output fell below the low cutoff limit for voltage output
CURRENT_MEASURE_HIGH	Clears cutoffs caused when the measured current exceeded the high cutoff limit for current output
CURRENT_MEASURE_LOW	Clears cutoffs caused when the measured current fell below the low cutoff limit for current output
VOLTAGE_CHANGE_HIGH	Clears cutoffs caused when the voltage slew rate increased beyond the positive change cutoff for voltage output
VOLTAGE_CHANGE_LOW	Clears cutoffs caused when the voltage slew rate decreased beyond the negative change cutoff for voltage output
CURRENT_CHANGE_HIGH	Clears cutoffs caused when the current slew rate increased beyond the positive change cutoff for current output
CURRENT_CHANGE_LOW	Clears cutoffs caused when the voltage slew rate decreased beyond the negative change cutoff for current output

close

`nidcpower.Session.`**`close`**`()`

Closes the session specified in **vi** and deallocates the resources that NI-DCPower reserves. If power output is enabled when you call this method, the output channels remain in their existing state and continue providing power. Use the `nidcpower.Session.ConfigureOutputEnabled()` method to disable power output on a per channel basis. Use the *`nidcpower.Session.reset()`* method to disable power output on all channel(s).

Related Topics:

Programming States

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

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

commit

`nidcpower.Session.`**`commit`**`()`

Applies previously configured settings to the specified channel(s). Calling this method moves the NI-DCPower session from the Uncommitted state into the Committed state. After calling this method, modifying any property reverts the NI-DCPower session to the Uncommitted state. Use the *`nidcpower.Session.initiate()`* method to transition to the Running state. Refer to the Programming States topic in the *NI DC Power Supplies and SMUs Help* for details about the specific NI-DCPower software states.

Related Topics:

Programming States

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].commit()`

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

Example: `my_session.commit()`

configure_aperture_time

`nidcpower.Session.`**`configure_aperture_time`**`(`*aperture_time, units=nidcpower.ApertureTimeUnits.SECONDS*`)`

Configures the aperture time on the specified channel(s).

The supported values depend on the **units**. Refer to the *Aperture Time* topic for your device in the *NI DC Power Supplies and SMUs Help* for more information. In general, devices support discrete **apertureTime** values, and if you configure **apertureTime** to some unsupported value, NI-DCPower coerces it up to the next supported value.

Refer to the *Measurement Configuration and Timing* or *DC Noise Rejection* topic for your device in the *NI DC Power Supplies and SMUs Help* for more information about how to configure your measurements.

Related Topics:

Aperture Time

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your `nidcpower.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_aperture_time()`

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

Example: `my_session.configure_aperture_time()`

Parameters

- **`aperture_time`** (*`float`*) – Specifies the aperture time. Refer to the *Aperture Time* topic for your device in the *NI DC Power Supplies and SMUs Help* for more information.
- **`units`** (*`nidcpower.ApertureTimeUnits`*) – Specifies the units for **apertureTime**. **Defined Values**:

SECONDS	Specifies seconds.
POWER_LINE_CYCLES	Specifies Power Line Cycles.

create_advanced_sequence

`nidcpower.Session.`**`create_advanced_sequence`**(*sequence_name*, *property_names*, *set_as_active_sequence=True*)

Creates an empty advanced sequence. Call the *`nidcpower.Session.create_advanced_sequence_step()`* method to add steps to the active advanced sequence.

You can create multiple advanced sequences in a session.

Support for this method

You must set the source mode to Sequence to use this method.

Using the *`nidcpower.Session.set_sequence()`* method with Advanced Sequence methods is unsupported.

Use this method in the Uncommitted or Committed programming states. Refer to the Programming States topic in the *NI DC Power Supplies and SMUs Help* for more information about NI-DCPower programming states.

Related Topics:

Advanced Sequence Mode

Programming States

`nidcpower.Session.create_advanced_sequence_step()`

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your *`nidcpower.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[ ... ].create_advanced_sequence()`

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

Example: `my_session.create_advanced_sequence()`

Parameters

- **`sequence_name`** (*`str`*) – Specifies the name of the sequence to create.
- **`property_names`** (*`list of str`*) – Specifies the names of the properties you reconfigure per step in the advanced sequence. The following table lists which properties can be configured in an advanced sequence for each NI-DCPower device that supports advanced sequencing. A Yes indicates that the property can be configured in advanced sequencing. An No indicates that the property cannot be configured in advanced sequencing.

Property	PXIe-4135	PXIe-4136	PXIe-4137	PXIe-4138
`nidcpower.Session.dc_noise_rejection`	Yes	No	Yes	No
`nidcpower.Session.aperture_time`	Yes	Yes	Yes	Yes

Table 1 – continued from previous page

Property	PXIe-4135	PXIe-4136	PXIe-4137	PXIe-4138
nidcpower.Session.measure_record_length	Yes	Yes	Yes	Yes
nidcpower.Session.sense	Yes	Yes	Yes	Yes
nidcpower.Session.ovp_enabled	Yes	Yes	Yes	No
nidcpower.Session.ovp_limit	Yes	Yes	Yes	No
nidcpower.Session.pulse_bias_delay	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_off_time	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_on_time	Yes	Yes	Yes	Yes
nidcpower.Session.source_delay	Yes	Yes	Yes	Yes
nidcpower.Session.current_compensation_frequency	Yes	No	Yes	No
nidcpower.Session.current_gain_bandwidth	Yes	No	Yes	No
nidcpower.Session.current_pole_zero_ratio	Yes	No	Yes	No
nidcpower.Session.voltage_compensation_frequency	Yes	No	Yes	No
nidcpower.Session.voltage_gain_bandwidth	Yes	No	Yes	No
nidcpower.Session.voltage_pole_zero_ratio	Yes	No	Yes	No
nidcpower.Session.current_level	Yes	Yes	Yes	Yes
nidcpower.Session.current_level_range	Yes	Yes	Yes	Yes
nidcpower.Session.voltage_limit	Yes	Yes	Yes	Yes
nidcpower.Session.voltage_limit_high	Yes	Yes	Yes	Yes
nidcpower.Session.voltage_limit_low	Yes	Yes	Yes	Yes
nidcpower.Session.voltage_limit_range	Yes	Yes	Yes	Yes
nidcpower.Session.current_limit	Yes	Yes	Yes	Yes
nidcpower.Session.current_limit_high	Yes	Yes	Yes	Yes
nidcpower.Session.current_limit_low	Yes	Yes	Yes	Yes
nidcpower.Session.current_limit_range	Yes	Yes	Yes	Yes
nidcpower.Session.voltage_level	Yes	Yes	Yes	Yes
nidcpower.Session.voltage_level_range	Yes	Yes	Yes	Yes
nidcpower.Session.output_enabled	Yes	Yes	Yes	Yes
nidcpower.Session.output_function	Yes	Yes	Yes	Yes
nidcpower.Session.output_resistance	Yes	No	Yes	No
nidcpower.Session.pulse_bias_current_level	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_bias_voltage_limit	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_bias_voltage_limit_high	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_bias_voltage_limit_low	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_current_level	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_current_level_range	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_voltage_limit	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_voltage_limit_high	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_voltage_limit_low	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_voltage_limit_range	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_bias_current_limit	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_bias_current_limit_high	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_bias_current_limit_low	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_bias_voltage_level	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_current_limit	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_current_limit_high	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_current_limit_low	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_current_limit_range	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_voltage_level	Yes	Yes	Yes	Yes
nidcpower.Session.pulse_voltage_level_range	Yes	Yes	Yes	Yes

Table 1 – continued from previous page

Property	PXIe-4135	PXIe-4136	PXIe-4137	PXIe-4138
nidcpower.Session.transient_response	Yes	Yes	Yes	Yes

- **set_as_active_sequence** (*bool*) – Specifies that this current sequence is active.

create_advanced_sequence_commit_step

nidcpower.Session.**create_advanced_sequence_commit_step**(*set_as_active_step=True*)

Creates a Commit step in the Active advanced sequence. A Commit step configures channels to a user-defined known state before starting the advanced sequence. When a Commit step exists in the Active advanced sequence, you cannot set the output method to Pulse Voltage or Pulse Current in either the Commit step (-1) or step 0. When you create an advanced sequence step, each property you passed to the *nidcpower.Session.create_advanced_sequence()* method is reset to its default value for that step unless otherwise specified.

Support for this Method

You must set the source mode to Sequence to use this method.

Using the *nidcpower.Session.set_sequence()* method with Advanced Sequence methods is unsupported.

Related Topics:

Advanced Sequence Mode

Programming States

nidcpower.Session.create_advanced_sequence()

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your *nidcpower.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[ ... ].create_advanced_sequence_commit_step()`

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

Example: `my_session.create_advanced_sequence_commit_step()`

Parameters **set_as_active_step** (*bool*) – Specifies whether the step created with this method is active in the Active advanced sequence.

create_advanced_sequence_step

nidcpower.Session.**create_advanced_sequence_step**(*set_as_active_step=True*)

Creates a new advanced sequence step in the advanced sequence specified by the Active advanced sequence. When you create an advanced sequence step, each property you passed to the *nidcpower.*

`Session.create_advanced_sequence()` method is reset to its default value for that step unless otherwise specified.

Support for this Method

You must set the source mode to Sequence to use this method.

Using the `nidcpower.Session.set_sequence()` method with Advanced Sequence methods is unsupported.

Related Topics:

Advanced Sequence Mode

Programming States

`nidcpower.Session.create_advanced_sequence()`

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].create_advanced_sequence_step()`

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

Example: `my_session.create_advanced_sequence_step()`

Parameters **`set_as_active_step`** (*`bool`*) – Specifies whether the step created with this method is active in the Active advanced sequence.

delete_advanced_sequence

`nidcpower.Session.`**`delete_advanced_sequence`**`(`*sequence_name*`)`

Deletes a previously created advanced sequence and all the advanced sequence steps in the advanced sequence.

Support for this Method

You must set the source mode to Sequence to use this method.

Using the `nidcpower.Session.set_sequence()` method with Advanced Sequence methods is unsupported.

Related Topics:

Advanced Sequence Mode

Programming States

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your *nidcpower.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[ ... ].delete_advanced_sequence()`

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

Example: `my_session.delete_advanced_sequence()`

Parameters **sequence_name** (*str*) – specifies the name of the sequence to delete.

disable

`nidcpower.Session.`**`disable`**`()`

This method performs the same actions as the *nidcpower.Session.reset()* method, except that this method also immediately sets the *nidcpower.Session.output_enabled* property to False.

This method opens the output relay on devices that have an output relay.

export_attribute_configuration_buffer

`nidcpower.Session.`**`export_attribute_configuration_buffer`**`()`

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

You can export and import session property configurations only between devices with identical model numbers and the same number of configured channels.

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

Support for this Method

Calling this method in Sequence Source Mode is unsupported.

Channel Mapping Behavior for Multichannel Sessions

When importing and exporting session property configurations between NI-DCPower sessions that were initialized with different channels, the configurations of the exporting channels are mapped to the importing channels in the order you specify in the **channelName** input to the `nidcpower.Session.__init__()` method.

For example, if your entry for **channelName** is 0,1 for the exporting session and 1,2 for the importing session:

- The configuration exported from channel 0 is imported into channel 1.
- The configuration exported from channel 1 is imported into channel 2.

Related Topics:

Using Properties and Properties

Setting Properties and Properties Before Reading Them

Note: This method will return an error if the total number of channels initialized for the exporting session is not equal to the total number of channels initialized for the importing session.

Return type bytes

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

export_attribute_configuration_file

`nidcpower.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 and the same number of configured channels.

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

Support for this Method

Calling this method in Sequence Source Mode is unsupported.

Channel Mapping Behavior for Multichannel Sessions

When importing and exporting session property configurations between NI-DCPower sessions that were initialized with different channels, the configurations of the exporting channels are mapped to the importing channels in the order you specify in the **channelName** input to the `nidcpower.Session.__init__()` method.

For example, if your entry for **channelName** is 0,1 for the exporting session and 1,2 for the importing session:

- The configuration exported from channel 0 is imported into channel 1.
- The configuration exported from channel 1 is imported into channel 2.

Related Topics:

Using Properties and Properties

Setting Properties and Properties Before Reading Them

Note: This method will return an error if the total number of channels initialized for the exporting session is not equal to the total number of channels initialized for the importing session.

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:** .nidcpowerconfig

fetch_multiple

`nidcpower.Session.`**`fetch_multiple`**`(`*count*, *timeout=hightime.timedelta(seconds=1.0)*`)`

Returns a list of named tuples (Measurement) that were previously taken and are stored in the NI-DCPower buffer. This method should not be used when the `nidcpower.Session.measure_when` property is set to `ON_DEMAND`. You must first call `nidcpower.Session.initiate()` before calling this method.

Fields in Measurement:

- **voltage** (float)
- **current** (float)
- **in_compliance** (bool)

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your `nidcpower.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_multiple()`

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

Example: `my_session.fetch_multiple()`

Parameters

- **`count`** (*`int`*) – Specifies the number of measurements to fetch.
- **`timeout`** (*`hightime.timedelta, datetime.timedelta, or float in seconds`*) – Specifies the maximum time allowed for this method to complete. If the method does not complete within this time interval, NI-DCPower returns an error.

  Note: When setting the timeout interval, ensure you take into account any triggers so that the timeout interval is long enough for your application.

Return type list of Measurement

Returns

List of named tuples with fields:

- **voltage** (float)
- **current** (float)
- **in_compliance** (bool)

get_channel_name

`nidcpower.Session.`**`get_channel_name`**`(`*index*`)`

Retrieves the output **channelName** that corresponds to the requested **index**. Use the `nidcpower.Session.channel_count` property to determine the upper bound of valid values for **index**.

Parameters **index** (*int*) – Specifies which output channel name to return. The index values begin at 1.

Return type str

Returns Returns the output channel name that corresponds to **index**.

get_channel_names

`nidcpower.Session.`**`get_channel_names`**`(`*indices*`)`

Returns a list of channel names for the given channel indices.

Parameters **indices** (*basic sequence types or 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_ext_cal_last_date_and_time

`nidcpower.Session.`**`get_ext_cal_last_date_and_time`**`()`

Returns the date and time of the last successful calibration.

Return type hightime.datetime

Returns Indicates date and time of the last calibration.

get_ext_cal_last_temp

`nidcpower.Session.`**`get_ext_cal_last_temp`**`()`

Returns the onboard **temperature** of the device, in degrees Celsius, during the last successful external calibration.

Return type float

Returns Returns the onboard **temperature** of the device, in degrees Celsius, during the last successful external calibration.

get_ext_cal_recommended_interval

`nidcpower.Session.`**`get_ext_cal_recommended_interval`**`()`

Returns the recommended maximum interval, in **months**, between external calibrations.

Return type hightime.timedelta

Returns Specifies the recommended maximum interval, in **months**, between external calibrations.

get_self_cal_last_date_and_time

`nidcpower.Session.`**`get_self_cal_last_date_and_time`**`()`

Returns the date and time of the oldest successful self-calibration from among the channels in the session.

Note: This method is not supported on all devices.

Return type hightime.datetime

Returns Returns the date and time the device was last calibrated.

get_self_cal_last_temp

`nidcpower.Session.`**`get_self_cal_last_temp`**`()`

Returns the onboard temperature of the device, in degrees Celsius, during the oldest successful self-calibration from among the channels in the session.

For example, if you have a session using channels 1 and 2, and you perform a self-calibration on channel 1 with a device temperature of 25 degrees Celsius at 2:00, and a self-calibration was performed on channel 2 at 27 degrees Celsius at 3:00 on the same day, this method returns 25 for the **temperature** parameter.

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Return type float

Returns Returns the onboard **temperature** of the device, in degrees Celsius, during the oldest successful calibration.

import_attribute_configuration_buffer

`nidcpower.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 and the same number of configured channels.

Support for this Method

Calling this method in Sequence Source Mode is unsupported.

Channel Mapping Behavior for Multichannel Sessions

When importing and exporting session property configurations between NI-DCPower sessions that were initialized with different channels, the configurations of the exporting channels are mapped to the importing channels in the order you specify in the **channelName** input to the `nidcpower.Session.__init__()` method.

For example, if your entry for **channelName** is 0,1 for the exporting session and 1,2 for the importing session:

- The configuration exported from channel 0 is imported into channel 1.
- The configuration exported from channel 1 is imported into channel 2.

Related Topics:

Programming States

Using Properties and Properties

Setting Properties and Properties Before Reading Them

Note: This method will return an error if the total number of channels initialized for the exporting session is not equal to the total number of channels initialized for the importing session.

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

import_attribute_configuration_file

`nidcpower.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 and the same number of configured channels.

Support for this Method

Calling this method in Sequence Source Mode is unsupported.

Channel Mapping Behavior for Multichannel Sessions

When importing and exporting session property configurations between NI-DCPower sessions that were initialized with different channels, the configurations of the exporting channels are mapped to the importing channels in the order you specify in the **channelName** input to the `nidcpower.Session.__init__()` method.

For example, if your entry for **channelName** is 0,1 for the exporting session and 1,2 for the importing session:

- The configuration exported from channel 0 is imported into channel 1.
- The configuration exported from channel 1 is imported into channel 2.

Related Topics:

Programming States

Using Properties and Properties

Setting Properties and Properties Before Reading Them

Note: This method will return an error if the total number of channels initialized for the exporting session is not equal to the total number of channels initialized for the importing session.

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:** .nidcpowerconfig

initiate

`nidcpower.Session.`**`initiate`**`()`

Starts generation or acquisition, causing the specified channel(s) to leave the Uncommitted state or Committed state and enter the Running state. To return to the Uncommitted state call the `nidcpower.Session.abort()` method. Refer to the Programming States topic in the *NI DC Power Supplies and SMUs Help* for information about the specific NI-DCPower software states.

Related Topics:

Programming States

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

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].initiate()`

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

Example: `my_session.initiate()`

lock

`nidcpower.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 `nidcpower.Session.lock()` method.
- A call to NI-DCPower locked the session.
- After a call to the `nidcpower.Session.lock()` method returns successfully, no other threads can access the device session until you call the `nidcpower.Session.unlock()` method or exit out of the with block when using lock context manager.
- Use the `nidcpower.Session.lock()` method and the `nidcpower.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 `nidcpower.Session.lock()` method within the same thread. To completely unlock the session, you must balance each call to the `nidcpower.Session.lock()` method with a call to the `nidcpower.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 nidcpower.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, `nidcpower.Session.lock()` acts as a context manager and unlock will be called when the *with* block is exited

measure

`nidcpower.Session.measure(measurement_type)`

Returns the measured value of either the voltage or current on the specified output channel. Each call to this method blocks other method calls until the hardware returns the **measurement**. To measure multiple output channels, use the `nidcpower.Session.measure_multiple()` method.

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].measure()`

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

Example: `my_session.measure()`

Parameters **measurement_type** (`nidcpower.MeasurementTypes`) – Specifies whether a voltage or current value is measured. **Defined Values**:

VOLTAGE	The device measures voltage.
CURRENT	The device measures current.

Return type float

Returns Returns the value of the measurement, either in volts for voltage or amps for current.

measure_multiple

`nidcpower.Session.measure_multiple()`

Returns a list of named tuples (Measurement) containing the measured voltage and current values on the specified output channel(s). Each call to this method blocks other method calls until the measurements are returned from the device. The order of the measurements returned in the array corresponds to the order on the specified output channel(s).

Fields in Measurement:

- **voltage** (float)
- **current** (float)

- **in_compliance** (bool) - Always None

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your *nidcpower.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[ ... ].measure_multiple()`

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

Example: `my_session.measure_multiple()`

Return type list of Measurement

Returns

List of named tuples with fields:

- **voltage** (float)
- **current** (float)
- **in_compliance** (bool) - Always None

query_in_compliance

`nidcpower.Session.`**`query_in_compliance`**`()`

Queries the specified output device to determine if it is operating at the compliance limit.

The compliance limit is the current limit when the output method is set to *DC_VOLTAGE*. If the output is operating at the compliance limit, the output reaches the current limit before the desired voltage level. Refer to the `nidcpower.Session.ConfigureOutputFunction()` method and the `nidcpower.Session.ConfigureCurrentLimit()` method for more information about output method and current limit, respectively.

The compliance limit is the voltage limit when the output method is set to *DC_CURRENT*. If the output is operating at the compliance limit, the output reaches the voltage limit before the desired current level. Refer to the `nidcpower.Session.ConfigureOutputFunction()` method and the `nidcpower.Session.ConfigureVoltageLimit()` method for more information about output method and voltage limit, respectively.

Related Topics:

Compliance

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

Tip: This method can be called on specific channels within your *nidcpower.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[ ... ].query_in_compliance()`

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

Example: `my_session.query_in_compliance()`

Return type bool

Returns Returns whether the device output channel is in compliance.

query_latched_output_cutoff_state

`nidcpower.Session.`**`query_latched_output_cutoff_state`**`(`*output_cutoff_reason*`)`

Discovers if an output cutoff limit was exceeded for the specified reason. When an output cutoff is engaged, the output of the channel(s) is disconnected. If a limit was exceeded, the state is latched until you clear it with the `nidcpower.Session.clear_latched_output_cutoff_state()` method or the `nidcpower.Session.reset()` method.

outputCutoffReason specifies the conditions for which an output is disconnected.

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].query_latched_output_cutoff_state()`

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

Example: `my_session.query_latched_output_cutoff_state()`

Parameters **`output_cutoff_reason`** (`nidcpower.OutputCutoffReason`) – Specifies which output cutoff conditions to query.

ALL	Any output cutoff condition was met
VOLTAGE_OUTPUT_HIGH	The output exceeded the high cutoff limit for voltage output
VOLTAGE_OUTPUT_LOW	The output fell below the low cutoff limit for voltage output
CURRENT_MEASURE_HIGH	The measured current exceeded the high cutoff limit for current output
CURRENT_MEASURE_LOW	The measured current fell below the low cutoff limit for current output
VOLTAGE_CHANGE_HIGH	The voltage slew rate increased beyond the positive change cutoff for voltage output
VOLTAGE_CHANGE_LOW	The voltage slew rate decreased beyond the negative change cutoff for voltage output
CURRENT_CHANGE_HIGH	The current slew rate increased beyond the positive change cutoff for current output
CURRENT_CHANGE_LOW	The current slew rate decreased beyond the negative change cutoff for current output

Return type bool

Returns

Specifies whether an output cutoff has engaged.

True	An output cutoff has engaged for the conditions in **output cutoff reason**.
False	No output cutoff has engaged.

query_max_current_limit

`nidcpower.Session.`**`query_max_current_limit`**`(`*voltage_level*`)`

Queries the maximum current limit on an output channel if the output channel is set to the specified **voltageLevel**.

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].query_max_current_limit()`

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

Example: `my_session.query_max_current_limit()`

Parameters **`voltage_level`** (`float`) – Specifies the voltage level to use when calculating the **maxCurrentLimit**.

Return type float

Returns Returns the maximum current limit that can be set with the specified **voltageLevel**.

query_max_voltage_level

`nidcpower.Session.`**`query_max_voltage_level`**`(`*current_limit*`)`

Queries the maximum voltage level on an output channel if the output channel is set to the specified **currentLimit**.

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].query_max_voltage_level()`

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

Example: `my_session.query_max_voltage_level()`

Parameters **`current_limit`** (`float`) – Specifies the current limit to use when calculating the **maxVoltageLevel**.

Return type float

Returns Returns the maximum voltage level that can be set on an output channel with the specified **currentLimit**.

query_min_current_limit

`nidcpower.Session.`**`query_min_current_limit`**`(`*voltage_level*`)`

Queries the minimum current limit on an output channel if the output channel is set to the specified **voltageLevel**.

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].query_min_current_limit()`

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

Example: `my_session.query_min_current_limit()`

Parameters **`voltage_level`** (`float`) – Specifies the voltage level to use when calculating the **minCurrentLimit**.

Return type float

Returns Returns the minimum current limit that can be set on an output channel with the specified **voltageLevel**.

query_output_state

`nidcpower.Session.`**`query_output_state`**`(`*output_state*`)`

Queries the specified output channel to determine if the output channel is currently in the state specified by **outputState**.

Related Topics:

Compliance

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].query_output_state()`

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

Example: `my_session.query_output_state()`

Parameters **`output_state`** (`nidcpower.OutputStates`) – Specifies the output state of the output channel that is being queried. **Defined Values**:

`VOLTAGE`	The device maintains a constant voltage by adjusting the current.
`CURRENT`	The device maintains a constant current by adjusting the voltage.

Return type bool

Returns Returns whether the device output channel is in the specified output state.

read_current_temperature

`nidcpower.Session.`**`read_current_temperature`**`()`

Returns the current onboard **temperature**, in degrees Celsius, of the device.

Return type float

Returns Returns the onboard **temperature**, in degrees Celsius, of the device.

reset

`nidcpower.Session.`**`reset`**`()`

Resets the specified channel(s) to a known state. This method disables power generation, resets session properties to their default values, commits the session properties, and leaves the session in the Uncommitted state. Refer to the Programming States topic for more information about NI-DCPower software states.

Tip: This method can be called on specific channels within your *`nidcpower.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[ ... ].reset()`

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

Example: `my_session.reset()`

reset_device

`nidcpower.Session.`**`reset_device`**`()`

Resets the device to a known state. The method disables power generation, resets session properties to their default values, clears errors such as overtemperature and unexpected loss of auxiliary power, commits the session properties, and leaves the session in the Uncommitted state. This method also performs a hard reset on the device and driver software. This method has the same functionality as using reset in Measurement & Automation Explorer. Refer to the Programming States topic for more information about NI-DCPower software states.

This will also open the output relay on devices that have an output relay.

reset_with_defaults

`nidcpower.Session.`**`reset_with_defaults`**`()`

Resets the device to a known state. This method disables power generation, resets session properties to their default values, commits the session properties, and leaves the session in the Running state. In addition to exhibiting the behavior of the *`nidcpower.Session.reset()`* method, this method can assign user-defined default values for configurable properties from the IVI configuration.

self_cal

`nidcpower.Session.`**`self_cal`**`()`

Performs a self-calibration upon the specified channel(s).

This method disables the output, performs several internal calculations, and updates calibration values. The updated calibration values are written to the device hardware if the `nidcpower.Session.self_calibration_persistence` property is set to `WRITE_TO_EEPROM`. Refer to the `nidcpower.Session.self_calibration_persistence` property topic for more information about the settings for this property.

When calling `nidcpower.Session.self_cal()` with the PXIe-4162/4163, specify all channels of your PXIe-4162/4163 with the channelName input. You cannot self-calibrate a subset of PXIe-4162/4163 channels.

Refer to the Self-Calibration topic for more information about this method.

Related Topics:

Self-Calibration

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your `nidcpower.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 `nidcpower.Session`.

Example: `my_session.self_cal()`

self_test

`nidcpower.Session.`**`self_test`**`()`

Performs the device self-test routine and returns the test result(s). Calling this method implicitly calls the `nidcpower.Session.reset()` method.

When calling `nidcpower.Session.self_test()` with the PXIe-4162/4163, specify all channels of your PXIe-4162/4163 with the channels input of `nidcpower.Session.__init__()`. You cannot self test a subset of PXIe-4162/4163 channels.

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	Self test passed.
1	Self test failed.

send_software_edge_trigger

`nidcpower.Session.`**`send_software_edge_trigger`**`(`*trigger*`)`

Asserts the specified trigger. This method can override an external edge trigger.

Related Topics:

Triggers

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your *nidcpower.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[ ... ].send_software_edge_trigger()`

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

Example: `my_session.send_software_edge_trigger()`

Parameters **`trigger`** (*nidcpower.SendSoftwareEdgeTriggerType*) – Specifies which trigger to assert. **Defined Values:**

NIDCPOWER_VAL_START_TRIGGER	Asserts the Start trigger.
NIDCPOWER_VAL_SOURCE_TRIGGER	Asserts the Source trigger.
NIDCPOWER_VAL_MEASURE_TRIGGER	Asserts the Measure trigger.
NIDCPOWER_VAL_SEQUENCE_ADVANCE_TRIGGER	Asserts the Sequence Advance trigger.
NIDCPOWER_VAL_PULSE_TRIGGER	Asserts the Pulse trigger.
NIDCPOWER_VAL_SHUTDOWN_TRIGGER	Asserts the Shutdown 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.

---

set_sequence

`nidcpower.Session.`**`set_sequence`**`(`*values*, *source_delays*`)`

Configures a series of voltage or current outputs and corresponding source delays. The source mode must be set to Sequence for this method to take effect.

Refer to the Configuring the Source Unit topic in the *NI DC Power Supplies and SMUs Help* for more information about how to configure your device.

Use this method in the Uncommitted or Committed programming states. Refer to the Programming States topic in the *NI DC Power Supplies and SMUs Help* for more information about NI-DCPower programming states.

Note: This method is not supported on all devices. Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].set_sequence()`

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

Example: `my_session.set_sequence()`

Parameters

- **values** (*list of float*) – Specifies the series of voltage levels or current levels, depending on the configured output method. **Valid values**: The valid values for this parameter are defined by the voltage level range or current level range.
- **source_delays** (*list of float*) – Specifies the source delay that follows the configuration of each value in the sequence. **Valid Values**: The valid values are between 0 and 167 seconds.

unlock

`nidcpower.Session.`**`unlock`**`()`

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

wait_for_event

`nidcpower.Session.`**`wait_for_event`**`(`*event_id*, *timeout=hightime.timedelta(seconds=10.0)*`)`

Waits until the specified channel(s) have generated the specified event.

The session monitors whether each type of event has occurred at least once since the last time this method or the `nidcpower.Session.initiate()` method were called. If an event has only been generated once and you call this method successively, the method times out. Individual events must be generated between separate calls of this method.

Note: Refer to Supported Methods by Device for more information about supported devices.

Tip: This method can be called on specific channels within your `nidcpower.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[ ... ].wait_for_event()`

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

Example: `my_session.wait_for_event()`

Parameters

- **event_id** (*nidcpower.Event*) – Specifies which event to wait for. **Defined Values:**

NIDCPOWER_VAL_SOURCE_COMPLETE_EVENT	Waits for the Source Complete event.
NIDCPOWER_VAL_MEASURE_COMPLETE_EVENT	Waits for the Measure Complete event.
NIDCPOWER_VAL_SEQUENCE_ITERATION_COMPLETE_EVENT	Waits for the Sequence Iteration Complete event.
NIDCPOWER_VAL_SEQUENCE_ENGINE_DONE_EVENT	Waits for the Sequence Engine Done event.
NIDCPOWER_VAL_PULSE_COMPLETE_EVENT	Waits for the Pulse Complete event.
NIDCPOWER_VAL_READY_FOR_PULSE_TRIGGER_EVENT	Waits for the Ready for Pulse Trigger event.

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.

- **timeout** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Specifies the maximum time allowed for this method to complete, in seconds. If the method does not complete within this time interval, NI-DCPower returns an error.

Note: When setting the timeout interval, ensure you take into account any triggers so that the timeout interval is long enough for your application.

Properties

active_advanced_sequence

nidcpower.Session.**active_advanced_sequence**

Specifies the advanced sequence to configure or generate.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.active_advanced_sequence`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Source:Advanced:Active Advanced Sequence**
- C Attribute: **NIDCPOWER_ATTR_ACTIVE_ADVANCED_SEQUENCE**

active_advanced_sequence_step

`nidcpower.Session.`**`active_advanced_sequence_step`**

Specifies the advanced sequence step to configure.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.active_advanced_sequence_step`

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: **Source:Advanced:Active Advanced Sequence Step**
- C Attribute: **NIDCPOWER_ATTR_ACTIVE_ADVANCED_SEQUENCE_STEP**

actual_power_allocation

`nidcpower.Session.`**`actual_power_allocation`**

Returns the power, in watts, the device is sourcing on each active channel if the `nidcpower.Session.power_allocation_mode` property is set to `AUTOMATIC` or `MANUAL`.

Valid Values: [0, device per-channel maximum power]

Default Value: Refer to the Supported Properties by Device topic for the default value by device.

Note: This property is not supported by all devices. Refer to the Supported Properties by Device topic for information about supported devices.

This property returns -1 when the *nidcpower.Session.power_allocation_mode* property is set to *DISABLED*.

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

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

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

Example: `my_session.actual_power_allocation`

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:

- LabVIEW Property: **Source:Advanced:Actual Power Allocation**
- C Attribute: **NIDCPOWER_ATTR_ACTUAL_POWER_ALLOCATION**

aperture_time

`nidcpower.Session.`**`aperture_time`**

Specifies the measurement aperture time for the channel configuration. Aperture time is specified in the units set by the *nidcpower.Session.aperture_time_units* property. for information about supported devices. Refer to the Aperture Time topic in the NI DC Power Supplies and SMUs Help for more information about how to configure your measurements and for information about valid values. Default Value: 0.01666666 seconds

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.aperture_time`

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: **Measurement:Aperture Time**
- C Attribute: **NIDCPOWER_ATTR_APERTURE_TIME**

aperture_time_units

`nidcpower.Session.`**`aperture_time_units`**

Specifies the units of the *nidcpower.Session.aperture_time* property for the channel configuration. for information about supported devices. Refer to the Aperture Time topic in the NI DC Power Supplies and SMUs Help for more information about how to configure your measurements and for information about valid values. Default Value: *SECONDS*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.aperture_time_units`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Aperture Time Units**
- C Attribute: **NIDCPOWER_ATTR_APERTURE_TIME_UNITS**

autorange

`nidcpower.Session.`**`autorange`**

Specifies whether the hardware automatically selects the best range to measure the signal. Note the highest range the algorithm uses is dependent on the corresponding limit range property. The algorithm the hardware uses can be controlled using the *`nidcpower.Session.autorange_aperture_time_mode`* property.

Note: Autoranging begins at module startup and remains active until the module is reconfigured or reset. This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.autorange`

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: **Measurement:Autorange**
- C Attribute: **NIDCPOWER_ATTR_AUTORANGE**

autorange_aperture_time_mode

`nidcpower.Session.`**`autorange_aperture_time_mode`**

Specifies whether the aperture time used for the measurement autorange algorithm is determined automatically or customized using the *`nidcpower.Session.autorange_minimum_aperture_time`* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.autorange_aperture_time_mode`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Advanced:Autorange Aperture Time Mode**
- C Attribute: **NIDCPOWER_ATTR_AUTORANGE_APERTURE_TIME_MODE**

autorange_behavior

`nidcpower.Session.`**`autorange_behavior`**

Specifies the algorithm the hardware uses for measurement autoranging.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.autorange_behavior`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Advanced:Autorange Behavior**
- C Attribute: **NIDCPOWER_ATTR_AUTORANGE_BEHAVIOR**

autorange_minimum_aperture_time

nidcpower.Session.**autorange_minimum_aperture_time**

Specifies the measurement autorange aperture time used for the measurement autorange algorithm. The aperture time is specified in the units set by the *nidcpower.Session.autorange_minimum_aperture_time_units* property. This value will typically be smaller than the aperture time used for measurements.

Note: For smaller ranges, the value is scaled up to account for noise. The factor used to scale the value is derived from the module capabilities. This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.autorange_minimum_aperture_time`

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: **Measurement:Advanced:Autorange Minimum Aperture Time**
- C Attribute: **NIDCPOWER_ATTR_AUTORANGE_MINIMUM_APERTURE_TIME**

autorange_minimum_aperture_time_units

nidcpower.Session.**autorange_minimum_aperture_time_units**

Specifies the units of the *nidcpower.Session.autorange_minimum_aperture_time* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.autorange_minimum_aperture_time_units`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Advanced:Autorange Minimum Aperture Time Units**
- C Attribute: **NIDCPOWER_ATTR_AUTORANGE_MINIMUM_APERTURE_TIME_UNITS**

autorange_minimum_current_range

`nidcpower.Session.`**`autorange_minimum_current_range`**

Specifies the lowest range used during measurement autoranging. Limiting the lowest range used during autoranging can improve the speed of the autoranging algorithm and minimize frequent and unpredictable range changes for noisy signals.

Note: The maximum range used is the range that includes the value specified in the compliance limit property, *nidcpower.Session.voltage_limit_range* property or *nidcpower.Session.current_limit_range* property, depending on the selected *nidcpower.Session.output_function*. This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.autorange_minimum_current_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: **Measurement:Advanced:Autorange Minimum Current Range**
- C Attribute: **NIDCPOWER_ATTR_AUTORANGE_MINIMUM_CURRENT_RANGE**

autorange_minimum_voltage_range

`nidcpower.Session.`**`autorange_minimum_voltage_range`**

Specifies the lowest range used during measurement autoranging. The maximum range used is range that includes the value specified in the compliance limit property. Limiting the lowest range used during autoranging can improve the speed of the autoranging algorithm and/or minimize thrashing between ranges for noisy signals.

Note: The maximum range used is the range that includes the value specified in the compliance limit property, *`nidcpower.Session.voltage_limit_range`* property or *`nidcpower.Session.current_limit_range`* property, depending on the selected *`nidcpower.Session.output_function`*. This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.autorange_minimum_voltage_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: **Measurement:Advanced:Autorange Minimum Voltage Range**
- C Attribute: **NIDCPOWER_ATTR_AUTORANGE_MINIMUM_VOLTAGE_RANGE**

autorange_threshold_mode

`nidcpower.Session.`**`autorange_threshold_mode`**

Specifies thresholds used during autoranging to determine when range changing occurs.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic.

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

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

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

Example: `my_session.autorange_threshold_mode`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Advanced:Autorange Threshold Mode**
- C Attribute: **NIDCPOWER_ATTR_AUTORANGE_THRESHOLD_MODE**

auto_zero

`nidcpower.Session.`**`auto_zero`**

Specifies the auto-zero method to use on the device. Refer to the NI PXI-4132 Measurement Configuration and Timing and Auto Zero topics for more information about how to configure your measurements. Default Value: The default value for the NI PXI-4132 is *`ON`*. The default value for all other devices is *`OFF`*, which is the only supported value for these devices.

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

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

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

Example: `my_session.auto_zero`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Auto Zero**

- C Attribute: **NIDCPOWER_ATTR_AUTO_ZERO**

auxiliary_power_source_available

`nidcpower.Session.`**`auxiliary_power_source_available`**

Indicates whether an auxiliary power source is connected to the device. A value of False may indicate that the auxiliary input fuse has blown. Refer to the Detecting Internal/Auxiliary Power topic in the NI DC Power Supplies and SMUs Help for more information about internal and auxiliary power. power source to generate power. Use the *`nidcpower.Session.power_source_in_use`* property to retrieve this information.

Note: This property does not necessarily indicate if the device is using the auxiliary

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: **Advanced:Auxiliary Power Source Available**
- C Attribute: **NIDCPOWER_ATTR_AUXILIARY_POWER_SOURCE_AVAILABLE**

channel_count

`nidcpower.Session.`**`channel_count`**

Indicates the number of channels that NI-DCPower supports for the instrument that was chosen when the current session was opened. 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: **NIDCPOWER_ATTR_CHANNEL_COUNT**

compliance_limit_symmetry

`nidcpower.Session.`**`compliance_limit_symmetry`**

Specifies whether compliance limits for current generation and voltage generation for the device are applied symmetrically about 0 V and 0 A or asymmetrically with respect to 0 V and 0 A. When set to **Symmetric**, voltage limits and current limits are set using a single property with a positive value. The resulting range is bounded by this positive value and its opposite. When set to **Asymmetric**, you must separately set a limit high and a limit low using distinct properties. For asymmetric limits, the range bounded by the limit high and limit low must include zero. **Default Value:** Symmetric **Related Topics:** Compliance Ranges Changing Ranges Overranging

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.compliance_limit_symmetry`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Source:Advanced:Compliance Limit Symmetry**
- C Attribute: **NIDCPOWER_ATTR_COMPLIANCE_LIMIT_SYMMETRY**

current_compensation_frequency

`nidcpower.Session.`**`current_compensation_frequency`**

The frequency at which a pole-zero pair is added to the system when the channel is in Constant Current mode. for information about supported devices. Default Value: Determined by the value of the *`NORMAL`* setting of the *`nidcpower.Session.transient_response`* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.current_compensation_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: **Source:Custom Transient Response:Current:Compensation Frequency**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_COMPENSATION_FREQUENCY**

current_gain_bandwidth

`nidcpower.Session.`**`current_gain_bandwidth`**

The frequency at which the unloaded loop gain extrapolates to 0 dB in the absence of additional poles and zeroes. This property takes effect when the channel is in Constant Current mode. for information about supported devices. Default Value: Determined by the value of the *NORMAL* setting of the *nidcpower.Session.transient_response* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.current_gain_bandwidth`

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: **Source:Custom Transient Response:Current:Gain Bandwidth**

- C Attribute: **NIDCPOWER_ATTR_CURRENT_GAIN_BANDWIDTH**

current_level

`nidcpower.Session.`**`current_level`**

Specifies the current level, in amps, that the device attempts to generate on the specified channel(s). This property is applicable only if the *nidcpower.Session.output_function* property is set to *DC_CURRENT*. *nidcpower.Session.output_enabled* property for more information about enabling the output channel. Valid Values: The valid values for this property are defined by the values to which the *nidcpower.Session.current_level_range* property is set.

Note: The channel must be enabled for the specified current level to take effect. Refer to the

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

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

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

Example: `my_session.current_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: **Source:DC Current:Current Level**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_LEVEL**

current_level_autorange

`nidcpower.Session.`**`current_level_autorange`**

Specifies whether NI-DCPower automatically selects the current level range based on the desired current level for the specified channels. If you set this property to *ON*, NI-DCPower ignores any changes you make to the *nidcpower.Session.current_level_range* property. If you change the *nidcpower.Session.current_level_autorange* property from *ON* to *OFF*, NI-DCPower retains the last value the *nidcpower.Session.current_level_range* property was set to (or the default value if the property was never set) and uses that value as the current level range. Query the *nidcpower.Session.current_level_range* property by using the `nidcpower.Session._get_attribute_vi_int32()` method for information about which range NI-DCPower automatically selects. The *nidcpower.Session.*

current_level_autorange property is applicable only if the *nidcpower.Session.output_function* property is set to *DC_CURRENT*. Default Value: *OFF*

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

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

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

Example: `my_session.current_level_autorange`

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: **Source:DC Current:Current Level Autorange**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_LEVEL_AUTORANGE**

---

current_level_range

`nidcpower.Session.`**`current_level_range`**

Specifies the current level range, in amps, for the specified channel(s). The range defines the valid value to which the current level can be set. Use the *nidcpower.Session.current_level_autorange* property to enable automatic selection of the current level range. The *nidcpower.Session.current_level_range* property is applicable only if the *nidcpower.Session.output_function* property is set to *DC_CURRENT*. *nidcpower.Session.output_enabled* property for more information about enabling the output channel. For valid ranges, refer to the Ranges topic for your device in the NI DC Power Supplies and SMUs Help.

Note: The channel must be enabled for the specified current level range to take effect. Refer to the

---

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

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

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

Example: `my_session.current_level_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: **Source:DC Current:Current Level Range**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_LEVEL_RANGE**

current_limit

`nidcpower.Session.`**`current_limit`**

Specifies the current limit, in amps, that the output cannot exceed when generating the desired voltage level on the specified channel(s). This property is applicable only if the *`nidcpower.Session.output_function`* property is set to *`DC_VOLTAGE`* and the *`nidcpower.Session.compliance_limit_symmetry`* property is set to *`SYMMETRIC`*. *`nidcpower.Session.output_enabled`* property for more information about enabling the output channel. Valid Values: The valid values for this property are defined by the values to which *`nidcpower.Session.current_limit_range`* property is set.

Note: The channel must be enabled for the specified current limit to take effect. Refer to the

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

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

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

Example: `my_session.current_limit`

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: **Source:DC Voltage:Current Limit**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_LIMIT**

current_limit_autorange

`nidcpower.Session.`**`current_limit_autorange`**

Specifies whether NI-DCPower automatically selects the current limit range based on the desired current limit for the specified channel(s). If you set this property to *ON*, NI-DCPower ignores any changes you make to the *nidcpower.Session.current_limit_range* property. If you change this property from *ON* to *OFF*, NI-DCPower retains the last value the *nidcpower.Session.current_limit_range* property was set to (or the default value if the property was never set) and uses that value as the current limit range. Query the *nidcpower.Session.current_limit_range* property by using the `nidcpower.Session._get_attribute_vi_int32()` method for information about which range NI-DCPower automatically selects. The *nidcpower.Session.current_limit_autorange* property is applicable only if the *nidcpower.Session.output_function* property is set to *DC_VOLTAGE*. Default Value: *OFF*

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

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

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

Example: `my_session.current_limit_autorange`

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: **Source:DC Voltage:Current Limit Autorange**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_LIMIT_AUTORANGE**

current_limit_behavior

`nidcpower.Session.`**`current_limit_behavior`**

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

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

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

Example: `my_session.current_limit_behavior`

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:

- C Attribute: **NIDCPOWER_ATTR_CURRENT_LIMIT_BEHAVIOR**

current_limit_high

`nidcpower.Session.`**`current_limit_high`**

Specifies the maximum current, in amps, that the output can produce when generating the desired voltage on the specified channel(s). This property is applicable only if the *Compliance Limit Symmetry <p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **DC Voltage**. You must also specify a *Current Limit Low <p:py:meth:'nidcpower.Session.CurrentLimitLow*.html>'__ to complete the asymmetric range. **Valid Values:** [1% of *Current Limit Range <p:py:meth:'nidcpower.Session.CurrentLimitRange*.html>'__, *Current Limit Range <p:py:meth:'nidcpower.Session.CurrentLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled <p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.current_limit_high`

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: **Source:DC Voltage:Current Limit High**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_LIMIT_HIGH**

current_limit_low

`nidcpower.Session.`**`current_limit_low`**

Specifies the minimum current, in amps, that the output can produce when generating the desired voltage on the specified channel(s). This property is applicable only if the *Compliance Limit Symmetry <p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **DC Voltage**. You must also specify a *Current Limit High <p:py:meth:'nidcpower.Session.CurrentLimitHigh*.html>'__ to complete the asymmetric range. **Valid Values:** [-*Current Limit Range <p:py:meth:'nidcpower.Session.CurrentLimitRange*.html>'__, -1% of *Current Limit Range <p:py:meth:'nidcpower.Session.CurrentLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled <p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.current_limit_low`

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: **Source:DC Voltage:Current Limit Low**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_LIMIT_LOW**

current_limit_range

`nidcpower.Session.`**`current_limit_range`**

Specifies the current limit range, in amps, for the specified channel(s). The range defines the valid value to which the current limit can be set. Use the *nidcpower.Session.current_limit_autorange* property to enable automatic selection of the current limit range. The *nidcpower.Session.current_limit_range* property is applicable only if the *nidcpower.Session.output_function* property is set to *DC_VOLTAGE*. *nidcpower.Session.output_enabled* property for more information about enabling the output channel. For valid ranges, refer to the Ranges topic for your device in the NI DC Power Supplies and SMUs Help.

Note: The channel must be enabled for the specified current limit to take effect. Refer to the

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

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

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

Example: `my_session.current_limit_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: **Source:DC Voltage:Current Limit Range**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_LIMIT_RANGE**

current_pole_zero_ratio

`nidcpower.Session.`**`current_pole_zero_ratio`**

The ratio of the pole frequency to the zero frequency when the channel is in Constant Current mode. for information about supported devices. Default Value: Determined by the value of the *NORMAL* setting of the *nidcpower.Session.transient_response* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

set.

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

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

Example: `my_session.current_pole_zero_ratio`

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: **Source:Custom Transient Response:Current:Pole-Zero Ratio**
- C Attribute: **NIDCPOWER_ATTR_CURRENT_POLE_ZERO_RATIO**

dc_noise_rejection

`nidcpower.Session.`**`dc_noise_rejection`**

Determines the relative weighting of samples in a measurement. Refer to the NI PXIe-4140/4141 DC Noise Rejection, NI PXIe-4142/4143 DC Noise Rejection, or NI PXIe-4144/4145 DC Noise Rejection topic in the NI DC Power Supplies and SMUs Help for more information about noise rejection. for information about supported devices. Default Value: *`NORMAL`*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.dc_noise_rejection`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Advanced:DC Noise Rejection**
- C Attribute: **NIDCPOWER_ATTR_DC_NOISE_REJECTION**

digital_edge_measure_trigger_input_terminal

`nidcpower.Session.`**`digital_edge_measure_trigger_input_terminal`**

Specifies the input terminal for the Measure trigger. This property is used only when the *`nidcpower.Session.measure_trigger_type`* property is set to *`DIGITAL_EDGE`*. for this property. You can specify any valid input terminal for this property. Valid terminals are listed in Measurement & Automation Explorer under the Device Routes tab. Input terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The input terminal can also be a terminal from another device. For example, you can set the input terminal on Dev1 to be /Dev2/SourceCompleteEvent.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.digital_edge_measure_trigger_input_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Measure Trigger:Digital Edge:Input Terminal**
- C Attribute: **NIDCPOWER_ATTR_DIGITAL_EDGE_MEASURE_TRIGGER_INPUT_TERMINAL**

digital_edge_pulse_trigger_input_terminal

`nidcpower.Session.`**`digital_edge_pulse_trigger_input_terminal`**

Specifies the input terminal for the Pulse trigger. This property is used only when the *`nidcpower.Session.pulse_trigger_type`* property is set to digital edge. You can specify any valid input terminal for this property. Valid terminals are listed in Measurement & Automation Explorer under the Device Routes tab. Input terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The input terminal can also be a terminal from another device. For example, you can set the input terminal on Dev1 to be /Dev2/SourceCompleteEvent.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.digital_edge_pulse_trigger_input_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Pulse Trigger:Digital Edge:Input Terminal**
- C Attribute: **NIDCPOWER_ATTR_DIGITAL_EDGE_PULSE_TRIGGER_INPUT_TERMINAL**

digital_edge_sequence_advance_trigger_input_terminal

`nidcpower.Session.`**`digital_edge_sequence_advance_trigger_input_terminal`**

Specifies the input terminal for the Sequence Advance trigger. Use this property only when the *nidcpower.Session.sequence_advance_trigger_type* property is set to *DIGITAL_EDGE*. the NI DC Power Supplies and SMUs Help for information about supported devices. You can specify any valid input terminal for this property. Valid terminals are listed in Measurement & Automation Explorer under the Device Routes tab. Input terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The input terminal can also be a terminal from another device. For example, you can set the input terminal on Dev1 to be /Dev2/SourceCompleteEvent.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic in

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

set.

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

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

Example: `my_session.digital_edge_sequence_advance_trigger_input_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Sequence Advance Trigger:Digital Edge:Input Terminal**
- C Attribute: **NIDCPOWER_ATTR_DIGITAL_EDGE_SEQUENCE_ADVANCE_TRIGGER_INPUT_TERMINA**

digital_edge_shutdown_trigger_input_terminal

`nidcpower.Session.`**`digital_edge_shutdown_trigger_input_terminal`**

Specifies the input terminal for the Shutdown trigger. This property is used only when the *`nidcpower.Session.shutdown_trigger_type`* property is set to digital edge. You can specify any valid input terminal for this property. Valid terminals are listed in Measurement & Automation Explorer under the Device Routes tab. Input terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The input terminal can also be a terminal from another device. For example, you can set the input terminal on Dev1 to be /Dev2/SourceCompleteEvent.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.digital_edge_shutdown_trigger_input_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Shutdown Trigger:Digital Edge:Input Terminal**
- C Attribute: **NIDCPOWER_ATTR_DIGITAL_EDGE_SHUTDOWN_TRIGGER_INPUT_TERMINAL**

digital_edge_source_trigger_input_terminal

`nidcpower.Session.`**`digital_edge_source_trigger_input_terminal`**

Specifies the input terminal for the Source trigger. Use this property only when the *`nidcpower.Session.source_trigger_type`* property is set to *`DIGITAL_EDGE`*. for information about supported devices. You can specify any valid input terminal for this property. Valid terminals are listed in Measurement & Automation Explorer under the Device Routes tab. Input terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The input terminal can also be a terminal from another device. For example, you can set the input terminal on Dev1 to be /Dev2/SourceCompleteEvent.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.digital_edge_source_trigger_input_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Source Trigger:Digital Edge:Input Terminal**
- C Attribute: **NIDCPOWER_ATTR_DIGITAL_EDGE_SOURCE_TRIGGER_INPUT_TERMINAL**

digital_edge_start_trigger_input_terminal

nidcpower.Session.**digital_edge_start_trigger_input_terminal**

Specifies the input terminal for the Start trigger. Use this property only when the *nidcpower.Session.start_trigger_type* property is set to *DIGITAL_EDGE*. for information about supported devices. You can specify any valid input terminal for this property. Valid terminals are listed in Measurement & Automation Explorer under the Device Routes tab. Input terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The input terminal can also be a terminal from another device. For example, you can set the input terminal on Dev1 to be /Dev2/SourceCompleteEvent.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.digital_edge_start_trigger_input_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Start Trigger:Digital Edge:Input Terminal**
- C Attribute: **NIDCPOWER_ATTR_DIGITAL_EDGE_START_TRIGGER_INPUT_TERMINAL**

driver_setup

nidcpower.Session.**driver_setup**

Indicates the Driver Setup string that you specified when initializing the driver. Some cases exist where you must specify the instrument driver options at initialization time. An example of this case is specifying a particular device model from among a family of devices that the driver supports. This property is useful when simulating a device. You can specify the driver-specific options through the DriverSetup keyword in the optionsString parameter in the `nidcpower.Session.__init__()` method or through the IVI Configuration Utility. You can specify driver-specific options through the DriverSetup keyword in the optionsString parameter in the `nidcpower.Session.__init__()` method. If you do not specify a Driver Setup string, this property returns an empty string.

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:Driver Setup**
- C Attribute: **NIDCPOWER_ATTR_DRIVER_SETUP**

exported_measure_trigger_output_terminal

`nidcpower.Session.`**`exported_measure_trigger_output_terminal`**

Specifies the output terminal for exporting the Measure trigger. Refer to the Device Routes tab in Measurement & Automation Explorer for a list of the terminals available on your device. for information about supported devices. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.exported_measure_trigger_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Measure Trigger:Export Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_EXPORTED_MEASURE_TRIGGER_OUTPUT_TERMINAL**

exported_pulse_trigger_output_terminal

`nidcpower.Session.`**`exported_pulse_trigger_output_terminal`**

Specifies the output terminal for exporting the Pulse trigger. Refer to the Device Routes tab in Measurement & Automation Explorer for a list of the terminals available on your device. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.exported_pulse_trigger_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Pulse Trigger:Export Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_EXPORTED_PULSE_TRIGGER_OUTPUT_TERMINAL**

exported_sequence_advance_trigger_output_terminal

`nidcpower.Session.`**`exported_sequence_advance_trigger_output_terminal`**

Specifies the output terminal for exporting the Sequence Advance trigger. Refer to the Device Routes tab in Measurement & Automation Explorer for a list of the terminals available on your device. for information about supported devices. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

set.

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

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

Example: `my_session.exported_sequence_advance_trigger_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Sequence Advance Trigger:Export Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_EXPORTED_SEQUENCE_ADVANCE_TRIGGER_OUTPUT_TERMINAL**

exported_source_trigger_output_terminal

`nidcpower.Session.`**`exported_source_trigger_output_terminal`**

Specifies the output terminal for exporting the Source trigger. Refer to the Device Routes tab in MAX for a list of the terminals available on your device. for information about supported devices. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.exported_source_trigger_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Source Trigger:Export Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_EXPORTED_SOURCE_TRIGGER_OUTPUT_TERMINAL**

exported_start_trigger_output_terminal

`nidcpower.Session.`**`exported_start_trigger_output_terminal`**

Specifies the output terminal for exporting the Start trigger. Refer to the Device Routes tab in Measurement & Automation Explorer (MAX) for a list of the terminals available on your device. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. for information about supported devices.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.exported_start_trigger_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Triggers:Start Trigger:Export Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_EXPORTED_START_TRIGGER_OUTPUT_TERMINAL**

fetch_backlog

`nidcpower.Session.`**`fetch_backlog`**

Returns the number of measurements acquired that have not been fetched yet.

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

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

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

Example: `my_session.fetch_backlog`

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: **Measurement:Fetch Backlog**
- C Attribute: **NIDCPOWER_ATTR_FETCH_BACKLOG**

instrument_firmware_revision

`nidcpower.Session.`**`instrument_firmware_revision`**

Contains the firmware revision information for the device you are currently using.

Tip: This property can be set/get on specific instruments within your *nidcpower.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 *nidcpower.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: **NIDCPOWER_ATTR_INSTRUMENT_FIRMWARE_REVISION**

instrument_manufacturer

`nidcpower.Session.`**`instrument_manufacturer`**

Contains the name of the manufacturer for the device you are currently using.

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

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

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

Example: `my_session.instrument_manufacturer`

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:Manufacturer**
- C Attribute: **NIDCPOWER_ATTR_INSTRUMENT_MANUFACTURER**

instrument_model

`nidcpower.Session.`**`instrument_model`**

Contains the model number or name of the device that you are currently using.

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

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

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

Example: `my_session.instrument_model`

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:Model**
- C Attribute: **NIDCPOWER_ATTR_INSTRUMENT_MODEL**

interlock_input_open

`nidcpower.Session.interlock_input_open`

Indicates whether the safety interlock circuit is open. Refer to the Safety Interlock topic in the NI DC Power Supplies and SMUs Help for more information about the safety interlock circuit. about supported devices.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information

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

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

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

Example: `my_session.interlock_input_open`

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: **Advanced:Interlock Input Open**
- C Attribute: **NIDCPOWER_ATTR_INTERLOCK_INPUT_OPEN**

io_resource_descriptor

`nidcpower.Session.io_resource_descriptor`

Indicates the resource descriptor NI-DCPower uses to identify the physical device. If you initialize NI-DCPower with a logical name, this property contains the resource descriptor that corresponds to the entry in the IVI Configuration utility. If you initialize NI-DCPower with the resource descriptor, this property contains that value.

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: **NIDCPOWER_ATTR_IO_RESOURCE_DESCRIPTOR**

logical_name

`nidcpower.Session.`**`logical_name`**

Contains the logical name you specified when opening the current IVI session. You can pass a logical name to the `nidcpower.Session.__init__()` method. The IVI Configuration utility must contain an entry for the logical name. The logical name entry refers to a method section in the IVI Configuration file. The method section specifies a physical device and initial user options.

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: **NIDCPOWER_ATTR_LOGICAL_NAME**

measure_buffer_size

`nidcpower.Session.`**`measure_buffer_size`**

Specifies the number of samples that the active channel measurement buffer can hold. The default value is the maximum number of samples that a device is capable of recording in one second. for information about supported devices. Valid Values: 1000 to 2147483647 Default Value: Varies by device. Refer to Supported Properties by Device topic in the NI DC Power Supplies and SMUs Help for more information about default values.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.measure_buffer_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: **Measurement:Advanced:Measure Buffer Size**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_BUFFER_SIZE**

measure_complete_event_delay

`nidcpower.Session.`**`measure_complete_event_delay`**

Specifies the amount of time to delay the generation of the Measure Complete event, in seconds. for information about supported devices. Valid Values: 0 to 167 seconds Default Value: The NI PXI-4132 and NI PXIe-4140/4141/4142/4143/4144/4145/4154 supports values from 0 seconds to 167 seconds.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.measure_complete_event_delay`

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: **Events:Measure Complete Event:Event Delay**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_COMPLETE_EVENT_DELAY**

measure_complete_event_output_terminal

`nidcpower.Session.`**`measure_complete_event_output_terminal`**

Specifies the output terminal for exporting the Measure Complete event. for information about supported devices. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.measure_complete_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Events:Measure Complete Event:Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_COMPLETE_EVENT_OUTPUT_TERMINAL**

measure_complete_event_pulse_polarity

`nidcpower.Session.`**`measure_complete_event_pulse_polarity`**

Specifies the behavior of the Measure Complete event. for information about supported devices. Default Value: *`HIGH`*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.measure_complete_event_pulse_polarity`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Events:Measure Complete Event:Pulse:Polarity**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_COMPLETE_EVENT_PULSE_POLARITY**

measure_complete_event_pulse_width

`nidcpower.Session.`**`measure_complete_event_pulse_width`**

Specifies the width of the Measure Complete event, in seconds. The minimum event pulse width value for PXI devices is 150 ns, and the minimum event pulse width value for PXI Express devices is 250 ns. The maximum event pulse width value for all devices is 1.6 microseconds. for information about supported devices. Valid Values: 1.5e-7 to 1.6e-6 Default Value: The default value for PXI devices is 150 ns. The default value for PXI Express devices is 250 ns.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.measure_complete_event_pulse_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: **Events:Measure Complete Event:Pulse:Width**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_COMPLETE_EVENT_PULSE_WIDTH**

measure_record_delta_time

`nidcpower.Session.`**`measure_record_delta_time`**

Queries the amount of time, in seconds, between between the start of two consecutive measurements in a measure record. Only query this property after the desired measurement settings are committed. for information about supported devices. two measurements and the rest would differ.

Note: This property is not available when Auto Zero is configured to Once because the amount of time between the first

---

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

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

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

Example: `my_session.measure_record_delta_time`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Measure Record Delta Time**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_RECORD_DELTA_TIME**

---

measure_record_length

`nidcpower.Session.`**`measure_record_length`**

Specifies how many measurements compose a measure record. When this property is set to a value greater than 1, the *`nidcpower.Session.measure_when`* property must be set to *`AUTOMATICALLY_AFTER_SOURCE_COMPLETE`* or *`ON_MEASURE_TRIGGER`*. for information about supported devices. Valid Values: 1 to 16,777,216 Default Value: 1

Note: This property is not available in a session involving multiple channels.

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

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

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

Example: `my_session.measure_record_length`

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: **Measurement:Measure Record Length**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_RECORD_LENGTH**

measure_record_length_is_finite

`nidcpower.Session.`**`measure_record_length_is_finite`**

Specifies whether to take continuous measurements. Call the *nidcpower.Session.abort()* method to stop continuous measurements. When this property is set to False and the *nidcpower.Session.source_mode* property is set to *SINGLE_POINT*, the *nidcpower.Session.measure_when* property must be set to *AUTOMATICALLY_AFTER_SOURCE_COMPLETE* or *ON_MEASURE_TRIGGER*. When this property is set to False and the *nidcpower.Session.source_mode* property is set to *SEQUENCE*, the *nidcpower.Session.measure_when* property must be set to *ON_MEASURE_TRIGGER*. for information about supported devices. Default Value: True

Note: This property is not available in a session involving multiple channels.

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

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

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

Example: `my_session.measure_record_length_is_finite`

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: **Measurement:Measure Record Length Is Finite**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_RECORD_LENGTH_IS_FINITE**

measure_trigger_type

nidcpower.Session.**measure_trigger_type**

Specifies the behavior of the Measure trigger. for information about supported devices. Default Value: *DIGITAL_EDGE*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.measure_trigger_type`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Triggers:Measure Trigger:Trigger Type**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_TRIGGER_TYPE**

measure_when

nidcpower.Session.**measure_when**

Specifies when the measure unit should acquire measurements. Unless this property is configured to *ON_MEASURE_TRIGGER*, the *nidcpower.Session.measure_trigger_type* property is ignored. Refer to the Acquiring Measurements topic in the NI DC Power Supplies and SMUs Help for more information about how to configure your measurements. Default Value: If the *nidcpower.Session.source_mode* property is set to *SINGLE_POINT*, the default value is *ON_DEMAND*. This value supports only the *nidcpower.Session.measure()* method and *nidcpower.Session.measure_multiple()* method. If the *nidcpower.Session.source_mode* property is set to *SEQUENCE*, the default value is *AUTOMATICALLY_AFTER_SOURCE_COMPLETE*. This value supports only the *nidcpower.Session.fetch_multiple()* method.

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

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

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

Example: `my_session.measure_when`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Advanced:Measure When**
- C Attribute: **NIDCPOWER_ATTR_MEASURE_WHEN**

merged_channels

`nidcpower.Session.`**`merged_channels`**

Specifies the channel(s) to merge with a designated primary channel of an SMU in order to increase the maximum current you can source from the SMU. This property designates the merge channels to combine with a primary channel. To designate the primary channel, initialize the session to the primary channel only. Note: You cannot change the merge configuration with this property when the session is in the Running state. For complete information on using merged channels with this property, refer to Merged Channels in the NI DC Power Supplies and SMUs Help.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices. Devices that do not support this property behave as if no channels were merged. Default Value: Refer to the Supported Properties by Device topic for the default value by device.

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

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

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

Example: `my_session.merged_channels`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Source:Advanced:Merged Channels**
- C Attribute: **NIDCPOWER_ATTR_MERGED_CHANNELS**

output_capacitance

`nidcpower.Session.`**`output_capacitance`**

Specifies whether to use a low or high capacitance on the output for the specified channel(s). for information about supported devices. Refer to the NI PXI-4130 Output Capacitance Selection topic in the NI DC Power Supplies and SMUs Help for more information about capacitance.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.output_capacitance`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Source:Advanced:Output Capacitance**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CAPACITANCE**

output_connected

`nidcpower.Session.`**`output_connected`**

Specifies whether the output relay is connected (closed) or disconnected (open). The `nidcpower.`

`Session.output_enabled` property does not change based on this property; they are independent of each other. about supported devices. Set this property to False to disconnect the output terminal from the output. to the output terminal might discharge unless the relay is disconnected. Excessive connecting and disconnecting of the output can cause premature wear on the relay. Default Value: True

Note: Only disconnect the output when disconnecting is necessary for your application. For example, a battery connected

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

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

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

Example: `my_session.output_connected`

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: **Source:Output Connected**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CONNECTED**

output_cutoff_current_change_limit_high

`nidcpower.Session.`**`output_cutoff_current_change_limit_high`**

Specifies a limit for positive current slew rate, in amps per microsecond, for output cutoff. If the current increases at a rate that exceeds this limit, the output is disconnected.

To find out whether an output has exceeded this limit, call the `nidcpower.Session.query_latched_output_cutoff_state()` method with `CURRENT_CHANGE_HIGH` as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_current_change_limit_high`

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: **Source:Output Cutoff:Current Change Limit High**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_CURRENT_CHANGE_LIMIT_HIGH**

output_cutoff_current_change_limit_low

`nidcpower.Session.`**`output_cutoff_current_change_limit_low`**

Specifies a limit for negative current slew rate, in amps per microsecond, for output cutoff. If the current decreases at a rate that exceeds this limit, the output is disconnected.

To find out whether an output has exceeded this limit, call the `nidcpower.Session.query_latched_output_cutoff_state()` method with `CURRENT_CHANGE_LOW` as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_current_change_limit_low`

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: **Source:Output Cutoff:Current Change Limit Low**

- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_CURRENT_CHANGE_LIMIT_LOW**

output_cutoff_current_measure_limit_high

`nidcpower.Session.`**`output_cutoff_current_measure_limit_high`**

Specifies a high limit current value, in amps, for output cutoff. If the measured current exceeds this limit, the output is disconnected.

To find out whether an output has exceeded this limit, call the `nidcpower.Session.query_latched_output_cutoff_state()` method with `CURRENT_MEASURE_HIGH` as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_current_measure_limit_high`

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: **Source:Output Cutoff:Current Measure Limit High**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_CURRENT_MEASURE_LIMIT_HIGH**

output_cutoff_current_measure_limit_low

`nidcpower.Session.`**`output_cutoff_current_measure_limit_low`**

Specifies a low limit current value, in amps, for output cutoff. If the measured current falls below this limit, the output is disconnected.

To find out whether an output has fallen below this limit, call the `nidcpower.Session.query_latched_output_cutoff_state()` method with `CURRENT_MEASURE_LOW` as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_current_measure_limit_low`

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: **Source:Output Cutoff:Current Measure Limit Low**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_CURRENT_MEASURE_LIMIT_LOW**

output_cutoff_current_overrange_enabled

`nidcpower.Session.`**`output_cutoff_current_overrange_enabled`**

Enables or disables current overrange functionality for output cutoff. If enabled, the output is disconnected when the measured current saturates the current range.

To find out whether an output has exceeded this limit, call the *nidcpower.Session.query_latched_output_cutoff_state()* method with *VOLTAGE_OUTPUT_HIGH* as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_current_overrange_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: **Source:Output Cutoff:Current Overrange Enabled**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_CURRENT_OVERRANGE_ENABLED**

output_cutoff_enabled

`nidcpower.Session.`**`output_cutoff_enabled`**

Enables or disables output cutoff functionality. If enabled, you can define output cutoffs that, if exceeded, cause the output of the specified channel(s) to be disconnected. When this property is disabled, all other output cutoff properties are ignored.

Note: Refer to Supported Properties by Device for information about supported devices. Instruments that do not support this property behave as if this property were set to False.

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

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

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

Example: `my_session.output_cutoff_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: **Source:Output Cutoff:Enabled**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_ENABLED**

output_cutoff_voltage_change_limit_high

`nidcpower.Session.`**`output_cutoff_voltage_change_limit_high`**

Specifies a limit for positive voltage slew rate, in volts per microsecond, for output cutoff. If the voltage increases at a rate that exceeds this limit, the output is disconnected.

To find out whether an output has exceeded this limit, call the *`nidcpower.Session.query_latched_output_cutoff_state()`* with *`VOLTAGE_CHANGE_HIGH`* as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_voltage_change_limit_high`

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: **Source:Output Cutoff:Voltage Change Limit High**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_VOLTAGE_CHANGE_LIMIT_HIGH**

output_cutoff_voltage_change_limit_low

`nidcpower.Session.`**`output_cutoff_voltage_change_limit_low`**

Specifies a limit for negative voltage slew rate, in volts per microsecond, for output cutoff. If the voltage decreases at a rate that exceeds this limit, the output is disconnected.

To find out whether an output has exceeded this limit, call the *`nidcpower.Session.query_latched_output_cutoff_state()`* with *`VOLTAGE_CHANGE_LOW`* as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_voltage_change_limit_low`

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: **Source:Output Cutoff:Voltage Change Limit Low**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_VOLTAGE_CHANGE_LIMIT_LOW**

output_cutoff_voltage_output_limit_high

`nidcpower.Session.`**`output_cutoff_voltage_output_limit_high`**

Specifies a high limit voltage value, in volts, for output cutoff. If the voltage output exceeds this limit, the output is disconnected.

To find out whether an output has exceeded this limit, call the *`nidcpower.Session.query_latched_output_cutoff_state()`* method with *`VOLTAGE_OUTPUT_HIGH`* as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_voltage_output_limit_high`

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: **Source:Output Cutoff:Voltage Output Limit High**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_VOLTAGE_OUTPUT_LIMIT_HIGH**

output_cutoff_voltage_output_limit_low

`nidcpower.Session.`**`output_cutoff_voltage_output_limit_low`**

Specifies a low limit voltage value, in volts, for output cutoff. If the voltage output falls below this limit, the output is disconnected.

To find out whether an output has fallen below this limit, call the *`nidcpower.Session.query_latched_output_cutoff_state()`* method with *`VOLTAGE_OUTPUT_LOW`* as the output cutoff reason.

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.output_cutoff_voltage_output_limit_low`

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: **Source:Output Cutoff:Voltage Output Limit Low**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_CUTOFF_VOLTAGE_OUTPUT_LIMIT_LOW**

output_enabled

`nidcpower.Session.`**`output_enabled`**

Specifies whether the output is enabled (True) or disabled (False). Depending on the value you specify for the *`nidcpower.Session.output_function`* property, you also must set the voltage level or current level in addition to enabling the output the *`nidcpower.Session.initiate()`* method. Refer to the Programming States topic in the NI DC Power Supplies and SMUs Help for more information about NI-DCPower programming states. Default Value: The default value is True if you use the `nidcpower.Session.__init__()` method to open the session. Otherwise the default value is False, including when you use a calibration session or the deprecated programming model.

Note: If the session is in the Committed or Uncommitted states, enabling the output does not take effect until you call

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

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

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

Example: `my_session.output_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: **Source:Output Enabled**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_ENABLED**

output_function

`nidcpower.Session.`**`output_function`**

Configures the method to generate on the specified channel(s). When *DC_VOLTAGE* is selected, the device generates the desired voltage level on the output as long as the output current is below the current limit. You can use the following properties to configure the channel when *DC_VOLTAGE* is selected: *nidcpower.Session.voltage_level* *nidcpower.Session.current_limit* *nidcpower.Session.current_limit_high* *nidcpower.Session.current_limit_low* *nidcpower.Session.voltage_level_range* *nidcpower.Session.current_limit_range* *nidcpower.Session.compliance_limit_symmetry* When *DC_CURRENT* is selected, the device generates the desired current level on the output as long as the output voltage is below the voltage limit. You can use the following properties to configure the channel when *DC_CURRENT* is selected: *nidcpower.Session.current_level* *nidcpower.Session.voltage_limit* *nidcpower.Session.voltage_limit_high* *nidcpower.Session.voltage_limit_low* *nidcpower.Session.current_level_range* *nidcpower.Session.voltage_limit_range* *nidcpower.Session.compliance_limit_symmetry*

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

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

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

Example: `my_session.output_function`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Source:Output Function**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_FUNCTION**

output_resistance

`nidcpower.Session.`**`output_resistance`**

Specifies the output resistance that the device attempts to generate for the specified channel(s). This property is available only when you set the *`nidcpower.Session.output_function`* property on a support device. Refer to a supported device's topic about output resistance for more information about selecting an output resistance. about supported devices. Default Value: 0.0

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic for information

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

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

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

Example: `my_session.output_resistance`

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: **Source:Output Resistance**
- C Attribute: **NIDCPOWER_ATTR_OUTPUT_RESISTANCE**

overranging_enabled

`nidcpower.Session.`**`overranging_enabled`**

Specifies whether NI-DCPower allows setting the voltage level, current level, voltage limit and current limit outside the device specification limits. True means that overranging is enabled. Refer to the Ranges topic in the NI DC Power Supplies and SMUs Help for more information about overranging. Default Value: False

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

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

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

Example: `my_session.overranging_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: **Source:Advanced:Overranging Enabled**
- C Attribute: **NIDCPOWER_ATTR_OVERRANGING_ENABLED**

ovp_enabled

`nidcpower.Session.`**`ovp_enabled`**

Enables (True) or disables (False) overvoltage protection (OVP). Refer to the Output Overvoltage Protection topic in the NI DC Power Supplies and SMUs Help for more information about overvoltage protection. for information about supported devices. Default Value: False

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.ovp_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: **Source:Advanced:OVP Enabled**
- C Attribute: **NIDCPOWER_ATTR_OVP_ENABLED**

ovp_limit

`nidcpower.Session.`**`ovp_limit`**

Determines the voltage limit, in volts, beyond which overvoltage protection (OVP) engages. for information about supported devices. Valid Values: 2 V to 210 V Default Value: 210 V

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.ovp_limit`

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: **Source:Advanced:OVP Limit**
- C Attribute: **NIDCPOWER_ATTR_OVP_LIMIT**

power_allocation_mode

`nidcpower.Session.`**`power_allocation_mode`**

Determines whether the device sources the power its source configuration requires or a specific wattage you request; determines whether NI-DCPower proactively checks that this sourcing power is within the maximum per-channel and overall sourcing power of the device.

When this property configures NI-DCPower to perform a sourcing power check, a device is not permitted to source power in excess of its maximum per-channel or overall sourcing power. If the check determines a source configuration or power request would require the device to do so, NI-DCPower returns an error.

When this property does not configure NI-DCPower to perform a sourcing power check, a device can attempt to fulfill source configurations that would require it to source power in excess of its maximum per-channel or overall sourcing power and may shut down to prevent damage.

Default Value: Refer to the Supported Properties by Device topic for the default value by device.

Note: This property is not supported by all devices. Refer to the Supported Properties by Device topic for information about supported devices. Devices that do not support this property behave as if this property were set to `DISABLED`.

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

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

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

Example: `my_session.power_allocation_mode`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Source:Advanced:Power Allocation Mode**
- C Attribute: **NIDCPOWER_ATTR_POWER_ALLOCATION_MODE**

power_line_frequency

`nidcpower.Session.`**`power_line_frequency`**

Specifies the power line frequency for specified channel(s). NI-DCPower uses this value to select a timebase for setting the `nidcpower.Session.aperture_time` property in power line cycles (PLCs). in the NI DC Power Supplies and SMUs Help for information about supported devices. Default Value: `NIDCPOWER_VAL_60_HERTZ`

Note: This property is not supported by all devices. Refer to the Supported Properties by Device topic

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 *nidcpower.Session* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.power_line_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: **Measurement:Power Line Frequency**
- C Attribute: **NIDCPOWER_ATTR_POWER_LINE_FREQUENCY**

power_source

`nidcpower.Session.`**`power_source`**

Specifies the power source to use. NI-DCPower switches the power source used by the device to the specified value. Default Value: *AUTOMATIC* is set to *AUTOMATIC*. However, if the session is in the Committed or Uncommitted state when you set this property, the power source selection only occurs after you call the *nidcpower.Session.initiate()* method.

Note: Automatic selection is not persistent and occurs only at the time this property

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Advanced:Power Source**
- C Attribute: **NIDCPOWER_ATTR_POWER_SOURCE**

power_source_in_use

`nidcpower.Session.`**`power_source_in_use`**

Indicates whether the device is using the internal or auxiliary power source to generate power.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.PowerSourceInUse
Permissions	read only
Repeated Capabilities	None

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

- LabVIEW Property: **Advanced:Power Source In Use**
- C Attribute: **NIDCPOWER_ATTR_POWER_SOURCE_IN_USE**

pulse_bias_current_level

`nidcpower.Session.`**`pulse_bias_current_level`**

Specifies the pulse bias current level, in amps, that the device attempts to generate on the specified channel(s) during the off phase of a pulse. This property is applicable only if the *`nidcpower.Session.output_function`* property is set to *`PULSE_CURRENT`*. Valid Values: The valid values for this property are defined by the values you specify for the *`nidcpower.Session.pulse_current_level_range`* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_bias_current_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: **Source:Pulse Current:Pulse Bias Current Level**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_CURRENT_LEVEL**

pulse_bias_current_limit

nidcpower.Session.**pulse_bias_current_limit**

Specifies the pulse bias current limit, in amps, that the output cannot exceed when generating the desired pulse bias voltage on the specified channel(s) during the off phase of a pulse. This property is applicable only if the *nidcpower.Session.output_function* property is set to *PULSE_VOLTAGE*. Valid Values: The valid values for this property are defined by the values you specify for the *nidcpower.Session.pulse_current_limit_range* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_bias_current_limit`

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: **Source:Pulse Voltage:Pulse Bias Current Limit**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_CURRENT_LIMIT**

pulse_bias_current_limit_high

nidcpower.Session.**pulse_bias_current_limit_high**

Specifies the maximum current, in amps, that the output can produce when generating the desired pulse voltage on the specified channel(s) during the *off* phase of a pulse. This property is applicable only if the *Compliance Limit Symmetry <p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **Pulse Voltage**. You must also specify a *Pulse Bias Current*

Limit Low <p:py:meth:'*nidcpower.Session.PulseBiasCurrentLimitLow*.html>'__ to complete the asymmetric range. **Valid Values:** [1% of *Pulse Current Limit Range* <p:py:meth:'*nidcpower.Session.PulseCurrentLimitRange*.html>'__, *Pulse Current Limit Range* <p:py:meth:'*nidcpower.Session.PulseCurrentLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

---

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled* <p:py:meth:'*nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE or if the *Output Method* <p:py:meth:'*nidcpower.Session.OutputFunction*.html>'__ property is set to a pulsing method.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.pulse_bias_current_limit_high`

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: **Source:Pulse Voltage:Pulse Bias Current Limit High**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_CURRENT_LIMIT_HIGH**

---

pulse_bias_current_limit_low

`nidcpower.Session.`**`pulse_bias_current_limit_low`**

Specifies the minimum current, in amps, that the output can produce when generating the desired pulse voltage on the specified channel(s) during the *off* phase of a pulse. This property is applicable only if the *Compliance Limit Symmetry* <p:py:meth:'*nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method* <p:py:meth:'*nidcpower.Session.OutputFunction*.html>'__ property is set to **Pulse Voltage**. You must also specify a *Pulse Bias Current Limit High* <p:py:meth:'*nidcpower.Session.PulseBiasCurrentLimitHigh*.html>'__ to complete the asymmetric range. **Valid Values:** [-*Pulse Current Limit Range* <p:py:meth:'*nidcpower.Session.PulseCurrentLimitRange*.html>'__, -1% of *Pulse Current Limit*

Range <p:py:meth:'nidcpower.Session.PulseCurrentLimitRange.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

---

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled <p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE or if the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to a pulsing method.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.pulse_bias_current_limit_low`

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: **Source:Pulse Voltage:Pulse Bias Current Limit Low**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_CURRENT_LIMIT_LOW**

---

pulse_bias_delay

`nidcpower.Session.`**`pulse_bias_delay`**

Determines when, in seconds, the device generates the Pulse Complete event after generating the off level of a pulse. Valid Values: 0 to 167 seconds Default Value: 16.67 milliseconds

---

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_bias_delay`

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: **Source:Advanced:Pulse Bias Delay**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_DELAY**

pulse_bias_voltage_level

`nidcpower.Session.`**`pulse_bias_voltage_level`**

Specifies the pulse bias voltage level, in volts, that the device attempts to generate on the specified channel(s) during the off phase of a pulse. This property is applicable only if the *nidcpower.Session.output_function* property is set to *PULSE_VOLTAGE*. Valid Values: The valid values for this property are defined by the values you specify for the *nidcpower.Session.pulse_voltage_level_range* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_bias_voltage_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: **Source:Pulse Voltage:Pulse Bias Voltage Level**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_VOLTAGE_LEVEL**

pulse_bias_voltage_limit

`nidcpower.Session.`**`pulse_bias_voltage_limit`**

Specifies the pulse voltage limit, in volts, that the output cannot exceed when generating the desired current on the specified channel(s) during the off phase of a pulse. This property is applicable only if the *`nidcpower.Session.output_function`* property is set to *`PULSE_CURRENT`*. Valid Values: The valid values for this property are defined by the values you specify for the *`nidcpower.Session.pulse_voltage_limit_range`* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_bias_voltage_limit`

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: **Source:Pulse Current:Pulse Bias Voltage Limit**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_VOLTAGE_LIMIT**

pulse_bias_voltage_limit_high

`nidcpower.Session.`**`pulse_bias_voltage_limit_high`**

Specifies the maximum voltage, in volts, that the output can produce when generating the desired pulse current on the specified channel(s) during the *off* phase of a pulse. This property is applicable only if the *Compliance Limit Symmetry <p:py:meth:‘nidcpower.Session.ComplianceLimitSymmetry*.html>‘__ property is set to **Asymmetric** and the *Output Method <p:py:meth:‘nidcpower.Session.OutputFunction*.html>‘__ property is set to **Pulse Current**. You must also specify a *Pulse Bias Voltage*

Limit Low <*p:py:meth:'nidcpower.Session.PulseBiasVoltageLimitLow*.html>'__ to complete the asymmetric range. **Valid Values:** [1% of *Pulse Voltage Limit Range* <*p:py:meth:'nidcpower.Session.PulseVoltageLimitRange*.html>'__, *Pulse Voltage Limit Range* <*p:py:meth:'nidcpower.Session.PulseVoltageLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled* <*p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE or if the *Output Method* <*p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to a pulsing method.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.pulse_bias_voltage_limit_high`

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: **Source:Pulse Current:Pulse Bias Voltage Limit High**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_VOLTAGE_LIMIT_HIGH**

pulse_bias_voltage_limit_low

`nidcpower.Session.`**`pulse_bias_voltage_limit_low`**

Specifies the minimum voltage, in volts, that the output can produce when generating the desired pulse current on the specified channel(s) during the *off* phase of a pulse. This property is applicable only if the *Compliance Limit Symmetry* <*p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method* <*p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **Pulse Current**. You must also specify a *Pulse Bias Voltage Limit High* <*p:py:meth:'nidcpower.Session.PulseBiasVoltageLimitHigh*.html>'__ to complete the asymmetric range. **Valid Values:** [-*Pulse Voltage Limit Range* <*p:py:meth:'nidcpower.Session.PulseVoltageLimitRange*.html>'__, -1% of *Pulse Voltage Limit*

Range <*p:py:meth:'nidcpower.Session.PulseVoltageLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

---

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled* <*p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE or if the *Output Method* <*p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to a pulsing method.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.pulse_bias_voltage_limit_low`

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: **Source:Pulse Current:Pulse Bias Voltage Limit Low**
- C Attribute: **NIDCPOWER_ATTR_PULSE_BIAS_VOLTAGE_LIMIT_LOW**

pulse_complete_event_output_terminal

`nidcpower.Session.`**`pulse_complete_event_output_terminal`**

Specifies the output terminal for exporting the Pulse Complete event. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. Default Value:The default value for PXI Express devices is 250 ns.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_complete_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Events:Pulse Complete Event:Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_PULSE_COMPLETE_EVENT_OUTPUT_TERMINAL**

pulse_complete_event_pulse_polarity

`nidcpower.Session.`**`pulse_complete_event_pulse_polarity`**

Specifies the behavior of the Pulse Complete event. Default Value: *HIGH*

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_complete_event_pulse_polarity`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Events:Pulse Complete Event:Pulse:Polarity**
- C Attribute: **NIDCPOWER_ATTR_PULSE_COMPLETE_EVENT_PULSE_POLARITY**

pulse_complete_event_pulse_width

`nidcpower.Session.`**`pulse_complete_event_pulse_width`**

Specifies the width of the Pulse Complete event, in seconds. The minimum event pulse width value for PXI Express devices is 250 ns. The maximum event pulse width value for PXI Express devices is 1.6 microseconds. Default Value: The default value for PXI Express devices is 250 ns.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_complete_event_pulse_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: **Events:Pulse Complete Event:Pulse:Width**
- C Attribute: **NIDCPOWER_ATTR_PULSE_COMPLETE_EVENT_PULSE_WIDTH**

pulse_current_level

`nidcpower.Session.`**`pulse_current_level`**

Specifies the pulse current level, in amps, that the device attempts to generate on the specified channel(s) during the on phase of a pulse. This property is applicable only if the *`nidcpower.Session.output_function`* property is set to *`PULSE_CURRENT`*. Valid Values: The valid values for this property are defined by the values you specify for the *`nidcpower.Session.pulse_current_level_range`* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_current_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: **Source:Pulse Current:Pulse Current Level**
- C Attribute: **NIDCPOWER_ATTR_PULSE_CURRENT_LEVEL**

pulse_current_level_range

`nidcpower.Session.`**`pulse_current_level_range`**

Specifies the pulse current level range, in amps, for the specified channel(s). The range defines the valid values to which you can set the pulse current level and pulse bias current level. This property is applicable only if the *nidcpower.Session.output_function* property is set to *PULSE_CURRENT*. For valid ranges, refer to the ranges topic for your device in the NI DC Power Supplies and SMUs Help.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_current_level_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: **Source:Pulse Current:Pulse Current Level Range**
- C Attribute: **NIDCPOWER_ATTR_PULSE_CURRENT_LEVEL_RANGE**

pulse_current_limit

`nidcpower.Session.`**`pulse_current_limit`**

Specifies the pulse current limit, in amps, that the output cannot exceed when generating the desired pulse voltage on the specified channel(s) during the on phase of a pulse. This property is applicable only if the *`nidcpower.Session.output_function`* property is set to *`PULSE_VOLTAGE`* and the *`nidcpower.Session.compliance_limit_symmetry`* property is set to *`SYMMETRIC`*. Valid Values: The valid values for this property are defined by the values you specify for the *`nidcpower.Session.pulse_current_limit_range`* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_current_limit`

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: **Source:Pulse Voltage:Pulse Current Limit**
- C Attribute: **NIDCPOWER_ATTR_PULSE_CURRENT_LIMIT**

pulse_current_limit_high

`nidcpower.Session.`**`pulse_current_limit_high`**

Specifies the maximum current, in amps, that the output can produce when generating the desired pulse voltage on the specified channel(s) during the *on* phase of a pulse. This property is applicable only if the *Compliance Limit Symmetry <p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **Pulse Voltage**. You must also specify a *Pulse Current Limit Low <p:py:meth:'nidcpower.Session.PulseCurrentLimitLow*.html>'__ to complete the asymmetric range. **Valid Values:** [1% of *Pulse Current Limit Range <p:py:meth:'nidcpower.Session.PulseCurrentLimitRange*.html>'__, *Pulse Current Limit Range <p:py:meth:'nidcpower.Session.PulseCurrentLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled <p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE or if the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to a pulsing method.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.pulse_current_limit_high`

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: **Source:Pulse Voltage:Pulse Current Limit High**
- C Attribute: **NIDCPOWER_ATTR_PULSE_CURRENT_LIMIT_HIGH**

pulse_current_limit_low

nidcpower.Session.**pulse_current_limit_low**

Specifies the minimum current, in amps, that the output can produce when generating the desired pulse voltage on the specified channel(s) during the *on* phase of a pulse. This property is applicable only if the *Compliance Limit Symmetry <p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **Pulse Voltage**. You must also specify a *Pulse Current Limit High <p:py:meth:'nidcpower.Session.PulseCurrentLimitHigh*.html>'__ to complete the asymmetric range. **Valid Values:** [-*Pulse Current Limit Range <p:py:meth:'nidcpower.Session.PulseCurrentLimitRange*.html>'__, -1% of *Pulse Current Limit Range <p:py:meth:'nidcpower.Session.PulseCurrentLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled <p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE or if the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to a pulsing method.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.pulse_current_limit_low`

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: **Source:Pulse Voltage:Pulse Current Limit Low**
- C Attribute: **NIDCPOWER_ATTR_PULSE_CURRENT_LIMIT_LOW**

pulse_current_limit_range

nidcpower.Session.**pulse_current_limit_range**

Specifies the pulse current limit range, in amps, for the specified channel(s). The range defines the valid values to which you can set the pulse current limit and pulse bias current limit. This property is applicable only if the *nidcpower.Session.output_function* property is set to *PULSE_VOLTAGE*. For valid ranges, refer to the ranges topic for your device in the NI DC Power Supplies and SMUs Help.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_current_limit_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: **Source:Pulse Voltage:Pulse Current Limit Range**
- C Attribute: **NIDCPOWER_ATTR_PULSE_CURRENT_LIMIT_RANGE**

pulse_off_time

nidcpower.Session.**pulse_off_time**

Determines the length, in seconds, of the off phase of a pulse. Valid Values: 10 microseconds to 167 seconds Default Value: 34 milliseconds

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_off_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: **Source:Advanced:Pulse Off Time**
- C Attribute: **NIDCPOWER_ATTR_PULSE_OFF_TIME**

pulse_on_time

`nidcpower.Session.`**`pulse_on_time`**

Determines the length, in seconds, of the on phase of a pulse. Valid Values: 10 microseconds to 167 seconds Default Value: 34 milliseconds

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_on_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: **Source:Advanced:Pulse On Time**
- C Attribute: **NIDCPOWER_ATTR_PULSE_ON_TIME**

pulse_trigger_type

nidcpower.Session.**pulse_trigger_type**

Specifies the behavior of the Pulse trigger. Default Value: *NONE*

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_trigger_type`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Triggers:Pulse Trigger:Trigger Type**
- C Attribute: **NIDCPOWER_ATTR_PULSE_TRIGGER_TYPE**

pulse_voltage_level

nidcpower.Session.**pulse_voltage_level**

Specifies the pulse current limit, in amps, that the output cannot exceed when generating the desired pulse voltage on the specified channel(s) during the on phase of a pulse. This property is applicable only if the *nidcpower.Session.output_function* property is set to *PULSE_VOLTAGE*. Valid Values: The valid values for this property are defined by the values you specify for the *nidcpower.Session.pulse_current_limit_range* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_voltage_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: **Source:Pulse Voltage:Pulse Voltage Level**
- C Attribute: **NIDCPOWER_ATTR_PULSE_VOLTAGE_LEVEL**

pulse_voltage_level_range

`nidcpower.Session.`**`pulse_voltage_level_range`**

Specifies the pulse voltage level range, in volts, for the specified channel(s). The range defines the valid values at which you can set the pulse voltage level and pulse bias voltage level. This property is applicable only if the *nidcpower.Session.output_function* property is set to *PULSE_VOLTAGE*. For valid ranges, refer to the ranges topic for your device in the NI DC Power Supplies and SMUs Help.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_voltage_level_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: **Source:Pulse Voltage:Pulse Voltage Level Range**

- C Attribute: **NIDCPOWER_ATTR_PULSE_VOLTAGE_LEVEL_RANGE**

pulse_voltage_limit

`nidcpower.Session.`**`pulse_voltage_limit`**

Specifies the pulse voltage limit, in volts, that the output cannot exceed when generating the desired pulse current on the specified channel(s) during the on phase of a pulse. This property is applicable only if the *`nidcpower.Session.output_function`* property is set to *`PULSE_CURRENT`* and the *`nidcpower.Session.compliance_limit_symmetry`* property is set to *`SYMMETRIC`*. Valid Values: The valid values for this property are defined by the values you specify for the *`nidcpower.Session.pulse_voltage_limit_range`* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.pulse_voltage_limit`

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: **Source:Pulse Current:Pulse Voltage Limit**
- C Attribute: **NIDCPOWER_ATTR_PULSE_VOLTAGE_LIMIT**

pulse_voltage_limit_high

`nidcpower.Session.`**`pulse_voltage_limit_high`**

Specifies the maximum voltage, in volts, that the output can produce when generating the desired pulse current on the specified channel(s) during the *on* phase of a pulse. This property is applicable only if the *Compliance Limit Symmetry <p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **Pulse Current**. You must also specify a *Pulse Voltage*

Limit Low <*p:py:meth:'nidcpower.Session.PulseVoltageLimitLow*.html>'__ to complete the asymmetric range. **Valid Values:** [1% of *Pulse Voltage Limit Range* <*p:py:meth:'nidcpower.Session.PulseVoltageLimitRange*.html>'__, *Pulse Voltage Limit Range* <*p:py:meth:'nidcpower.Session.PulseVoltageLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled* <*p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE or if the *Output Method* <*p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to a pulsing method.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.pulse_voltage_limit_high`

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: **Source:Pulse Current:Pulse Voltage Limit High**
- C Attribute: **NIDCPOWER_ATTR_PULSE_VOLTAGE_LIMIT_HIGH**

pulse_voltage_limit_low

`nidcpower.Session.`**`pulse_voltage_limit_low`**

Specifies the minimum voltage, in volts, that the output can produce when generating the desired pulse current on the specified channel(s) during the *on* phase of a pulse. This property is applicable only if the *Compliance Limit Symmetry* <*p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method* <*p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **Pulse Current**. You must also specify a *Pulse Voltage Limit High* <*p:py:meth:'nidcpower.Session.PulseVoltageLimitHigh*.html>'__ to complete the asymmetric range. **Valid Values:** [-*Pulse Voltage Limit Range* <*p:py:meth:'nidcpower.Session.PulseVoltageLimitRange*.html>'__, -1% of *Pulse Voltage Limit*

Range <p:py:meth:'nidcpower.Session.PulseVoltageLimitRange.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

---

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled <p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE or if the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to a pulsing method.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.pulse_voltage_limit_low`

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: **Source:Pulse Current:Pulse Voltage Limit Low**
- C Attribute: **NIDCPOWER_ATTR_PULSE_VOLTAGE_LIMIT_LOW**

pulse_voltage_limit_range

`nidcpower.Session.`**`pulse_voltage_limit_range`**

Specifies the pulse voltage limit range, in volts, for the specified channel(s). The range defines the valid values to which you can set the pulse voltage limit and pulse bias voltage limit. This property is applicable only if the `nidcpower.Session.output_function` property is set to `PULSE_CURRENT`. For valid ranges, refer to the ranges topic for your device in the NI DC Power Supplies and SMUs Help.

Note: The channel must be enabled for the specified current limit to take effect. Refer to the `nidcpower.Session.output_enabled` property for more information about enabling the output channel.

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

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

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

Example: `my_session.pulse_voltage_limit_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: **Source:Pulse Current:Pulse Voltage Limit Range**
- C Attribute: **NIDCPOWER_ATTR_PULSE_VOLTAGE_LIMIT_RANGE**

query_instrument_status

`nidcpower.Session.`**`query_instrument_status`**

Specifies whether NI-DCPower queries the device status after each operation. Querying the device status is useful for debugging. After you validate your program, you can set this property to False to disable status checking and maximize performance. NI-DCPower ignores status checking for particular properties regardless of the setting of this property. Use the `nidcpower.Session.__init__()` method to override this value. Default Value: True

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:Query Instrument Status**
- C Attribute: **NIDCPOWER_ATTR_QUERY_INSTRUMENT_STATUS**

ready_for_pulse_trigger_event_output_terminal

`nidcpower.Session.`**`ready_for_pulse_trigger_event_output_terminal`**

Specifies the output terminal for exporting the Ready For Pulse Trigger event. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.ready_for_pulse_trigger_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Events:Ready For Pulse Trigger Event:Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_READY_FOR_PULSE_TRIGGER_EVENT_OUTPUT_TERMINAL**

ready_for_pulse_trigger_event_pulse_polarity

`nidcpower.Session.`**`ready_for_pulse_trigger_event_pulse_polarity`**

Specifies the behavior of the Ready For Pulse Trigger event. Default Value: *`HIGH`*

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.ready_for_pulse_trigger_event_pulse_polarity`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Events:Ready For Pulse Trigger Event:Pulse:Polarity**
- C Attribute: **NIDCPOWER_ATTR_READY_FOR_PULSE_TRIGGER_EVENT_PULSE_POLARITY**

ready_for_pulse_trigger_event_pulse_width

`nidcpower.Session.`**`ready_for_pulse_trigger_event_pulse_width`**

Specifies the width of the Ready For Pulse Trigger event, in seconds. The minimum event pulse width value for PXI Express devices is 250 ns. The maximum event pulse width value for all devices is 1.6 microseconds. Default Value: The default value for PXI Express devices is 250 ns

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.ready_for_pulse_trigger_event_pulse_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: **Events:Ready For Pulse Trigger Event:Pulse:Width**
- C Attribute: **NIDCPOWER_ATTR_READY_FOR_PULSE_TRIGGER_EVENT_PULSE_WIDTH**

requested_power_allocation

`nidcpower.Session.`**`requested_power_allocation`**

Specifies the power, in watts, to request the device to source from each active channel. This property defines the power to source from the device only if the *nidcpower.Session.power_allocation_mode* property is set to *MANUAL*.

The power you request with this property may be incompatible with the power a given source configuration requires or the power the device can provide: If the requested power is less than the power required for the source configuration, the device does not exceed the requested power, and NI-DCPower returns an error. If the requested power is greater than the maximum per-channel or overall sourcing power, the device does not exceed the allowed power, and NI-DCPower returns an error.

Valid Values: [0, device per-channel maximum power] Default Value: Refer to the Supported Properties by Device topic for the default value by device.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic for information about supported devices.

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

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

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

Example: `my_session.requested_power_allocation`

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: **Source:Advanced:Requested Power Allocation**
- C Attribute: **NIDCPOWER_ATTR_REQUESTED_POWER_ALLOCATION**

---

reset_average_before_measurement

`nidcpower.Session.`**`reset_average_before_measurement`**

Specifies whether the measurement returned from any measurement call starts with a new measurement call (True) or returns a measurement that has already begun or completed(False). for information about supported devices. When you set the *nidcpower.Session.samples_to_average* property in the Running state, the output channel measurements might move out of synchronization. While NI-DCPower automatically synchronizes measurements upon the initialization of a session, you can force a synchronization in the running state before you run the `nidcpower.Session.measure_multiple()` method. To force a synchronization in the running state, set this property to True, and then run the `nidcpower.Session.measure_multiple()` method, specifying all channels in the channel name parameter. You can set the `nidcpower.Session.reset_average_before_measurement` property to False after the `nidcpower.Session.measure_multiple()` method completes. Default Value: True

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.reset_average_before_measurement`

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: **Measurement:Advanced:Reset Average Before Measurement**
- C Attribute: **NIDCPOWER_ATTR_RESET_AVERAGE_BEFORE_MEASUREMENT**

samples_to_average

nidcpower.Session.**samples_to_average**

Specifies the number of samples to average when you take a measurement. Increasing the number of samples to average decreases measurement noise but increases the time required to take a measurement. Refer to the NI PXI-4110, NI PXI-4130, NI PXI-4132, or NI PXIe-4154 Averaging topic for optional property settings to improve immunity to certain noise types, or refer to the NI PXIe-4140/4141 DC Noise Rejection, NI PXIe-4142/4143 DC Noise Rejection, or NI PXIe-4144/4145 DC Noise Rejection topic for information about improving noise immunity for those devices. Default Value: NI PXI-4110 or NI PXI-4130—10 NI PXI-4132—1 NI PXIe-4112—1 NI PXIe-4113—1 NI PXIe-4140/4141—1 NI PXIe-4142/4143—1 NI PXIe-4144/4145—1 NI PXIe-4154—500

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

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

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

Example: `my_session.samples_to_average`

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: **Measurement:Samples To Average**
- C Attribute: **NIDCPOWER_ATTR_SAMPLES_TO_AVERAGE**

self_calibration_persistence

nidcpower.Session.**self_calibration_persistence**

Specifies whether the values calculated during self-calibration should be written to hardware to be used until the next self-calibration or only used until the `nidcpower.Session.reset_device()` method is called or the machine is powered down. This property affects the behavior of the `nidcpower.Session.self_cal()` method. When set to `KEEP_IN_MEMORY`, the values calculated by the `nidcpower.Session.self_cal()` method are used in the existing session, as well as in all further sessions until you call the `nidcpower.Session.reset_device()` method or restart the machine. When you set this property to `WRITE_TO_EEPROM`, the values calculated by the `nidcpower.Session.self_cal()` method are written to hardware and used in the existing session and in all subsequent sessions until another call to the `nidcpower.Session.self_cal()` method is made. about supported devices. Default Value: `KEEP_IN_MEMORY`

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information

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

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

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

Example: `my_session.self_calibration_persistence`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.SelfCalibrationPersistence
Permissions	read-write
Repeated Capabilities	instruments

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

- LabVIEW Property: **Advanced:Self-Calibration Persistence**
- C Attribute: **NIDCPOWER_ATTR_SELF_CALIBRATION_PERSISTENCE**

sense

`nidcpower.Session.`**`sense`**

Selects either local or remote sensing of the output voltage for the specified channel(s). Refer to the Local and Remote Sense topic in the NI DC Power Supplies and SMUs Help for more information about sensing voltage on supported channels and about devices that support local and/or remote sensing. Default Value: The default value is *`LOCAL`* if the device supports local sense. Otherwise, the default and only supported value is *`REMOTE`*.

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

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

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

Example: `my_session.sense`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Measurement:Sense**
- C Attribute: **NIDCPOWER_ATTR_SENSE**

sequence_advance_trigger_type

`nidcpower.Session.`**`sequence_advance_trigger_type`**

Specifies the behavior of the Sequence Advance trigger. for information about supported devices. Default Value: *`NONE`*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.sequence_advance_trigger_type`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Triggers:Sequence Advance Trigger:Trigger Type**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_ADVANCE_TRIGGER_TYPE**

sequence_engine_done_event_output_terminal

`nidcpower.Session.`**`sequence_engine_done_event_output_terminal`**

Specifies the output terminal for exporting the Sequence Engine Done Complete event. for information about supported devices. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.sequence_engine_done_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Events:Sequence Engine Done Event:Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_ENGINE_DONE_EVENT_OUTPUT_TERMINAL**

sequence_engine_done_event_pulse_polarity

`nidcpower.Session.`**`sequence_engine_done_event_pulse_polarity`**

Specifies the behavior of the Sequence Engine Done event. for information about supported devices. Default Value: *`HIGH`*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.sequence_engine_done_event_pulse_polarity`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Events:Sequence Engine Done Event:Pulse:Polarity**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_ENGINE_DONE_EVENT_PULSE_POLARITY**

sequence_engine_done_event_pulse_width

`nidcpower.Session.`**`sequence_engine_done_event_pulse_width`**

Specifies the width of the Sequence Engine Done event, in seconds. The minimum event pulse width value for PXI devices is 150 ns, and the minimum event pulse width value for PXI Express devices is

250 ns. The maximum event pulse width value for all devices is 1.6 microseconds. for information about supported devices. Valid Values: 1.5e-7 to 1.6e-6 seconds Default Value: The default value for PXI devices is 150 ns. The default value for PXI Express devices is 250 ns.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.sequence_engine_done_event_pulse_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: **Events:Sequence Engine Done Event:Pulse:Width**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_ENGINE_DONE_EVENT_PULSE_WIDTH**

sequence_iteration_complete_event_output_terminal

`nidcpower.Session.`**`sequence_iteration_complete_event_output_terminal`**

Specifies the output terminal for exporting the Sequence Iteration Complete event. for information about supported devices. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.sequence_iteration_complete_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Events:Sequence Iteration Complete Event:Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_ITERATION_COMPLETE_EVENT_OUTPUT_TERMINAL**

sequence_iteration_complete_event_pulse_polarity

`nidcpower.Session.`**`sequence_iteration_complete_event_pulse_polarity`**

Specifies the behavior of the Sequence Iteration Complete event. for information about supported devices. Default Value: *`HIGH`*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.sequence_iteration_complete_event_pulse_polarity`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Events:Sequence Iteration Complete Event:Pulse:Polarity**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_ITERATION_COMPLETE_EVENT_PULSE_POLARITY**

sequence_iteration_complete_event_pulse_width

`nidcpower.Session.`**`sequence_iteration_complete_event_pulse_width`**

Specifies the width of the Sequence Iteration Complete event, in seconds. The minimum event pulse width value for PXI devices is 150 ns, and the minimum event pulse width value for PXI Express devices is 250 ns. The maximum event pulse width value for all devices is 1.6 microseconds. the NI DC Power Supplies and SMUs Help for information about supported devices. Valid Values: 1.5e-7 to 1.6e-6 seconds Default Value: The default value for PXI devices is 150 ns. The default value for PXI Express devices is 250 ns.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic in

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

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

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

Example: `my_session.sequence_iteration_complete_event_pulse_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: **Events:Sequence Iteration Complete Event:Pulse:Width**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_ITERATION_COMPLETE_EVENT_PULSE_WIDTH**

sequence_loop_count

`nidcpower.Session.`**`sequence_loop_count`**

Specifies the number of times a sequence is run after initiation. Refer to the Sequence Source Mode topic in the NI DC Power Supplies and SMUs Help for more information about the sequence loop count. for information about supported devices. When the *nidcpower.Session.sequence_loop_count_is_finite* property is set to False, the *nidcpower.Session.sequence_loop_count* property is ignored. Valid Range: 1 to 134217727 Default Value: 1

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.sequence_loop_count`

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: **Source:Advanced:Sequence Loop Count**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_LOOP_COUNT**

sequence_loop_count_is_finite

`nidcpower.Session.`**`sequence_loop_count_is_finite`**

Specifies whether a sequence should repeat indefinitely. Refer to the Sequence Source Mode topic in the NI DC Power Supplies and SMUs Help for more information about infinite sequencing. *nidcpower.Session.sequence_loop_count_is_finite* property is set to False, the *nidcpower.Session.sequence_loop_count* property is ignored. Default Value: True

Note: This property is not supported by all devices. When the

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

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

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

Example: `my_session.sequence_loop_count_is_finite`

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: **Source:Advanced:Sequence Loop Count Is Finite**
- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_LOOP_COUNT_IS_FINITE**

sequence_step_delta_time

`nidcpower.Session.`**`sequence_step_delta_time`**

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

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

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

Example: `my_session.sequence_step_delta_time`

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:

- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_STEP_DELTA_TIME**

---

sequence_step_delta_time_enabled

`nidcpower.Session.`**`sequence_step_delta_time_enabled`**

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

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

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

Example: `my_session.sequence_step_delta_time_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:

- C Attribute: **NIDCPOWER_ATTR_SEQUENCE_STEP_DELTA_TIME_ENABLED**

serial_number

`nidcpower.Session.`**`serial_number`**

Contains the serial number for the device you are currently using.

Tip: This property can be set/get on specific instruments within your *`nidcpower.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 *`nidcpower.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: **Inherent IVI Attributes:Instrument Identification:Serial Number**
- C Attribute: **NIDCPOWER_ATTR_SERIAL_NUMBER**

shutdown_trigger_type

`nidcpower.Session.`**`shutdown_trigger_type`**

Specifies the behavior of the Shutdown trigger. Default Value: *`NONE`*

Note: This property is not supported by all devices. Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.shutdown_trigger_type`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Triggers:Shutdown Trigger:Trigger Type**
- C Attribute: **NIDCPOWER_ATTR_SHUTDOWN_TRIGGER_TYPE**

simulate

`nidcpower.Session.`**`simulate`**

Specifies whether to simulate NI-DCPower I/O operations. True specifies that operation is simulated. Default Value: 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: **Inherent IVI Attributes:User Options:Simulate**
- C Attribute: **NIDCPOWER_ATTR_SIMULATE**

source_complete_event_output_terminal

`nidcpower.Session.`**`source_complete_event_output_terminal`**

Specifies the output terminal for exporting the Source Complete event. for information about supported devices. Output terminals can be specified in one of two ways. If the device is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.source_complete_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	channels

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

- LabVIEW Property: **Events:Source Complete Event:Output Terminal**
- C Attribute: **NIDCPOWER_ATTR_SOURCE_COMPLETE_EVENT_OUTPUT_TERMINAL**

source_complete_event_pulse_polarity

`nidcpower.Session.`**`source_complete_event_pulse_polarity`**

Specifies the behavior of the Source Complete event. for information about supported devices. Default Value: *`HIGH`*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.source_complete_event_pulse_polarity`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Events:Source Complete Event:Pulse:Polarity**
- C Attribute: **NIDCPOWER_ATTR_SOURCE_COMPLETE_EVENT_PULSE_POLARITY**

source_complete_event_pulse_width

nidcpower.Session.**source_complete_event_pulse_width**

Specifies the width of the Source Complete event, in seconds. for information about supported devices. The minimum event pulse width value for PXI devices is 150 ns, and the minimum event pulse width value for PXI Express devices is 250 ns. The maximum event pulse width value for all devices is 1.6 microseconds Valid Values: 1.5e-7 to 1.6e-6 seconds Default Value: The default value for PXI devices is 150 ns. The default value for PXI Express devices is 250 ns.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.source_complete_event_pulse_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: **Events:Source Complete Event:Pulse:Width**
- C Attribute: **NIDCPOWER_ATTR_SOURCE_COMPLETE_EVENT_PULSE_WIDTH**

source_delay

nidcpower.Session.**source_delay**

Determines when, in seconds, the device generates the Source Complete event, potentially starting a measurement if the *nidcpower.Session.measure_when* property is set to *AUTOMATICALLY_AFTER_SOURCE_COMPLETE*. Refer to the Single Point Source Mode and Sequence Source Mode topics for more information. Valid Values: 0 to 167 seconds Default Value: 0.01667 seconds

Note: Refer to Supported Properties by Device for information about supported devices.

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

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

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

Example: `my_session.source_delay`

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: **Source:Advanced:Source Delay**
- C Attribute: **NIDCPOWER_ATTR_SOURCE_DELAY**

source_mode

`nidcpower.Session.`**`source_mode`**

Specifies whether to run a single output point or a sequence. Refer to the Single Point Source Mode and Sequence Source Mode topics in the NI DC Power Supplies and SMUs Help for more information about source modes. Default value: *`SINGLE_POINT`*

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

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

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

Example: `my_session.source_mode`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Source:Source Mode**
- C Attribute: **NIDCPOWER_ATTR_SOURCE_MODE**

source_trigger_type

`nidcpower.Session.`**`source_trigger_type`**

Specifies the behavior of the Source trigger. for information about supported devices. Default Value: *NONE*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.source_trigger_type`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Triggers:Source Trigger:Trigger Type**
- C Attribute: **NIDCPOWER_ATTR_SOURCE_TRIGGER_TYPE**

specific_driver_description

`nidcpower.Session.`**`specific_driver_description`**

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: **NIDCPOWER_ATTR_SPECIFIC_DRIVER_DESCRIPTION**

specific_driver_prefix

nidcpower.Session.**specific_driver_prefix**

Contains the prefix for NI-DCPower. The name of each user-callable method in NI-DCPower begins with this prefix.

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 Prefix**
- C Attribute: **NIDCPOWER_ATTR_SPECIFIC_DRIVER_PREFIX**

specific_driver_revision

nidcpower.Session.**specific_driver_revision**

Contains additional version information about NI-DCPower.

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: **NIDCPOWER_ATTR_SPECIFIC_DRIVER_REVISION**

specific_driver_vendor

nidcpower.Session.**specific_driver_vendor**

Contains the name of the vendor that supplies NI-DCPower.

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: **NIDCPOWER_ATTR_SPECIFIC_DRIVER_VENDOR**

start_trigger_type

`nidcpower.Session.`**`start_trigger_type`**

Specifies the behavior of the Start trigger. for information about supported devices. Default Value: *NONE*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.start_trigger_type`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Triggers:Start Trigger:Trigger Type**
- C Attribute: **NIDCPOWER_ATTR_START_TRIGGER_TYPE**

supported_instrument_models

`nidcpower.Session.`**`supported_instrument_models`**

Contains a comma-separated (,) list of supported NI-DCPower device models.

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: **NIDCPOWER_ATTR_SUPPORTED_INSTRUMENT_MODELS**

transient_response

nidcpower.Session.**transient_response**

Specifies the transient response. Refer to the Transient Response topic in the NI DC Power Supplies and SMUs Help for more information about transient response. for information about supported devices. Default Value: *NORMAL*

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.transient_response`

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Source:Transient Response**
- C Attribute: **NIDCPOWER_ATTR_TRANSIENT_RESPONSE**

voltage_compensation_frequency

nidcpower.Session.**voltage_compensation_frequency**

The frequency at which a pole-zero pair is added to the system when the channel is in Constant Voltage mode. for information about supported devices. Default value: Determined by the value of the *NORMAL* setting of the *nidcpower.Session.transient_response* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.voltage_compensation_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: **Source:Custom Transient Response:Voltage:Compensation Frequency**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_COMPENSATION_FREQUENCY**

voltage_gain_bandwidth

`nidcpower.Session.`**`voltage_gain_bandwidth`**

The frequency at which the unloaded loop gain extrapolates to 0 dB in the absence of additional poles and zeroes. This property takes effect when the channel is in Constant Voltage mode. for information about supported devices. Default Value: Determined by the value of the *NORMAL* setting of the *nidcpower.Session.transient_response* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.voltage_gain_bandwidth`

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: **Source:Custom Transient Response:Voltage:Gain Bandwidth**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_GAIN_BANDWIDTH**

voltage_level

nidcpower.Session.**voltage_level**

Specifies the voltage level, in volts, that the device attempts to generate on the specified channel(s). This property is applicable only if the *nidcpower.Session.output_function* property is set to *DC_VOLTAGE*. *nidcpower.Session.output_enabled* property for more information about enabling the output channel. Valid Values: The valid values for this property are defined by the values you specify for the *nidcpower.Session.voltage_level_range* property.

Note: The channel must be enabled for the specified voltage level to take effect. Refer to the

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

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

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

Example: `my_session.voltage_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: **Source:DC Voltage:Voltage Level**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_LEVEL**

voltage_level_autorange

nidcpower.Session.**voltage_level_autorange**

Specifies whether NI-DCPower automatically selects the voltage level range based on the desired voltage level for the specified channel(s). If you set this property to *ON*, NI-DCPower ignores any changes you make to the *nidcpower.Session.voltage_level_range* property. If you change the *nidcpower.Session.voltage_level_autorange* property from *ON* to *OFF*, NI-DCPower retains the last value the *nidcpower.Session.voltage_level_range* property was set to (or the default value if the property was never set) and uses that value as the voltage level range. Query the `nidcpower.Session.voltage_level_range` property by using the `nidcpower.Session._get_attribute_vi_int32()` method for information about which range NI-DCPower automatically selects. The `nidcpower.Session.voltage_level_autorange` property is applicable only if the `nidcpower.Session.output_function` property is set to `DC_VOLTAGE`. Default Value: `OFF`

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

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

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

Example: `my_session.voltage_level_autorange`

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: **Source:DC Voltage:Voltage Level Autorange**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_LEVEL_AUTORANGE**

---

voltage_level_range

`nidcpower.Session.`**`voltage_level_range`**

Specifies the voltage level range, in volts, for the specified channel(s). The range defines the valid values to which the voltage level can be set. Use the `nidcpower.Session.voltage_level_autorange` property to enable automatic selection of the voltage level range. The `nidcpower.Session.voltage_level_range` property is applicable only if the `nidcpower.Session.output_function` property is set to `DC_VOLTAGE`. `nidcpower.Session.output_enabled` property for more information about enabling the output channel. For valid ranges, refer to the Ranges topic for your device in the NI DC Power Supplies and SMUs Help.

Note: The channel must be enabled for the specified voltage level range to take effect. Refer to the

---

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

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

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

Example: `my_session.voltage_level_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: **Source:DC Voltage:Voltage Level Range**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_LEVEL_RANGE**

voltage_limit

`nidcpower.Session.`**`voltage_limit`**

Specifies the voltage limit, in volts, that the output cannot exceed when generating the desired current level on the specified channels. This property is applicable only if the *`nidcpower.Session.output_function`* property is set to *`DC_CURRENT`* and the *`nidcpower.Session.compliance_limit_symmetry`* property is set to *`SYMMETRIC`*. *`nidcpower.Session.output_enabled`* property for more information about enabling the output channel. Valid Values: The valid values for this property are defined by the values to which the *`nidcpower.Session.voltage_limit_range`* property is set.

Note: The channel must be enabled for the specified current level to take effect. Refer to the

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

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

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

Example: `my_session.voltage_limit`

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: **Source:DC Current:Voltage Limit**

- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_LIMIT**

voltage_limit_autorange

`nidcpower.Session.`**`voltage_limit_autorange`**

Specifies whether NI-DCPower automatically selects the voltage limit range based on the desired voltage limit for the specified channel(s). If this property is set to *`ON`*, NI-DCPower ignores any changes you make to the *`nidcpower.Session.voltage_limit_range`* property. If you change the *`nidcpower.Session.voltage_limit_autorange`* property from *`ON`* to *`OFF`*, NI-DCPower retains the last value the *`nidcpower.Session.voltage_limit_range`* property was set to (or the default value if the property was never set) and uses that value as the voltage limit range. Query the *`nidcpower.Session.voltage_limit_range`* property by using the `nidcpower.Session._get_attribute_vi_int32()` method to find out which range NI-DCPower automatically selects. The *`nidcpower.Session.voltage_limit_autorange`* property is applicable only if the *`nidcpower.Session.output_function`* property is set to *`DC_CURRENT`*. Default Value: *`OFF`*

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

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

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

Example: `my_session.voltage_limit_autorange`

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: **Source:DC Current:Voltage Limit Autorange**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_LIMIT_AUTORANGE**

voltage_limit_high

`nidcpower.Session.`**`voltage_limit_high`**

Specifies the maximum voltage, in volts, that the output can produce when generating the desired current on the specified channel(s). This property is applicable only if the *Compliance Limit Symmetry <p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method <p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **DC Current**. You must also specify a *Voltage Limit Low <p:py:meth:'nidcpower.Session.VoltageLimitLow*.html>'__ to complete the asymmetric range. **Valid Values:** [1% of *Voltage Limit Range*

<*p:py:meth:'nidcpower.Session.VoltageLimitRange*.html>'__, *Voltage Limit Range* <*p:py:meth:'nidcpower.Session.VoltageLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled* <*p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE.

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 *nidcpower.Session* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.voltage_limit_high`

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: **Source:DC Current:Voltage Limit High**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_LIMIT_HIGH**

voltage_limit_low

`nidcpower.Session.`**`voltage_limit_low`**

Specifies the minimum voltage, in volts, that the output can produce when generating the desired current on the specified channel(s). This property is applicable only if the *Compliance Limit Symmetry* <*p:py:meth:'nidcpower.Session.ComplianceLimitSymmetry*.html>'__ property is set to **Asymmetric** and the *Output Method* <*p:py:meth:'nidcpower.Session.OutputFunction*.html>'__ property is set to **DC Current**. You must also specify a *Voltage Limit High* <*p:py:meth:'nidcpower.Session.VoltageLimitHigh*.html>'__ to complete the asymmetric range. **Valid Values:** [-*Voltage Limit Range* <*p:py:meth:'nidcpower.Session.VoltageLimitRange*.html>'__, -1% of *Voltage Limit Range* <*p:py:meth:'nidcpower.Session.VoltageLimitRange*.html>'__] The range bounded by the limit high and limit low must include zero. **Default Value:** Refer to Supported Properties by Device for the default value by device. **Related Topics:** Ranges Changing Ranges Overranging

Note: The limit may be extended beyond the selected limit range if the *Overranging Enabled* <*p:py:meth:'nidcpower.Session.OverrangingEnabled*.html>'__ property is set to TRUE.

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 `nidcpower.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

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

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

Example: `my_session.voltage_limit_low`

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: **Source:DC Current:Voltage Limit Low**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_LIMIT_LOW**

voltage_limit_range

`nidcpower.Session.`**`voltage_limit_range`**

Specifies the voltage limit range, in volts, for the specified channel(s). The range defines the valid values to which the voltage limit can be set. Use the `nidcpower.Session.voltage_limit_autorange` property to enable automatic selection of the voltage limit range. The `nidcpower.Session.voltage_limit_range` property is applicable only if the `nidcpower.Session.output_function` property is set to `DC_CURRENT`. `nidcpower.Session.output_enabled` property for more information about enabling the output channel. For valid ranges, refer to the Ranges topic for your device in the NI DC Power Supplies and SMUs Help.

Note: The channel must be enabled for the specified voltage limit range to take effect. Refer to the

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

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

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

Example: `my_session.voltage_limit_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: **Source:DC Current:Voltage Limit Range**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_LIMIT_RANGE**

voltage_pole_zero_ratio

nidcpower.Session.**voltage_pole_zero_ratio**

The ratio of the pole frequency to the zero frequency when the channel is in Constant Voltage mode. for information about supported devices. Default value: Determined by the value of the *NORMAL* setting of the *nidcpower.Session.transient_response* property.

Note: This property is not supported by all devices. Refer to Supported Properties by Device topic

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

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

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

Example: `my_session.voltage_pole_zero_ratio`

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: **Source:Custom Transient Response:Voltage:Pole-Zero Ratio**
- C Attribute: **NIDCPOWER_ATTR_VOLTAGE_POLE_ZERO_RATIO**

Session

- *merged_channels*
- *output_capacitance*
- *output_connected*
- *output_cutoff_current_change_limit_high*
- *output_cutoff_current_change_limit_low*
- *output_cutoff_current_measure_limit_high*
- *output_cutoff_current_measure_limit_low*
- *output_cutoff_current_overrange_enabled*
- *output_cutoff_enabled*
- *output_cutoff_voltage_change_limit_high*
- *output_cutoff_voltage_change_limit_low*
- *output_cutoff_voltage_output_limit_high*
- *output_cutoff_voltage_output_limit_low*
- *output_enabled*
- *output_function*
- *output_resistance*
- *overranging_enabled*
- *ovp_enabled*
- *ovp_limit*
- *power_allocation_mode*
- *power_line_frequency*
- *power_source*
- *power_source_in_use*
- *pulse_bias_current_level*
- *pulse_bias_current_limit*
- *pulse_bias_current_limit_high*
- *pulse_bias_current_limit_low*
- *pulse_bias_delay*
- *pulse_bias_voltage_level*
- *pulse_bias_voltage_limit*
- *pulse_bias_voltage_limit_high*
- *pulse_bias_voltage_limit_low*
- *pulse_complete_event_output_terminal*
- *pulse_complete_event_pulse_polarity*
- *pulse_complete_event_pulse_width*

- *pulse_current_level*
- *pulse_current_level_range*
- *pulse_current_limit*
- *pulse_current_limit_high*
- *pulse_current_limit_low*
- *pulse_current_limit_range*
- *pulse_off_time*
- *pulse_on_time*
- *pulse_trigger_type*
- *pulse_voltage_level*
- *pulse_voltage_level_range*
- *pulse_voltage_limit*
- *pulse_voltage_limit_high*
- *pulse_voltage_limit_low*
- *pulse_voltage_limit_range*
- *query_instrument_status*
- *ready_for_pulse_trigger_event_output_terminal*
- *ready_for_pulse_trigger_event_pulse_polarity*
- *ready_for_pulse_trigger_event_pulse_width*
- *requested_power_allocation*
- *reset_average_before_measurement*
- *samples_to_average*
- *self_calibration_persistence*
- *sense*
- *sequence_advance_trigger_type*
- *sequence_engine_done_event_output_terminal*
- *sequence_engine_done_event_pulse_polarity*
- *sequence_engine_done_event_pulse_width*
- *sequence_iteration_complete_event_output_terminal*
- *sequence_iteration_complete_event_pulse_polarity*
- *sequence_iteration_complete_event_pulse_width*
- *sequence_loop_count*
- *sequence_loop_count_is_finite*
- *sequence_step_delta_time*
- *sequence_step_delta_time_enabled*

- *serial_number*
- *shutdown_trigger_type*
- *simulate*
- *source_complete_event_output_terminal*
- *source_complete_event_pulse_polarity*
- *source_complete_event_pulse_width*
- *source_delay*
- *source_mode*
- *source_trigger_type*
- *specific_driver_description*
- *specific_driver_prefix*
- *specific_driver_revision*
- *specific_driver_vendor*
- *start_trigger_type*
- *supported_instrument_models*
- *transient_response*
- *voltage_compensation_frequency*
- *voltage_gain_bandwidth*
- *voltage_level*
- *voltage_level_autorange*
- *voltage_level_range*
- *voltage_limit*
- *voltage_limit_autorange*
- *voltage_limit_high*
- *voltage_limit_low*
- *voltage_limit_range*
- *voltage_pole_zero_ratio*

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 `niDCPower_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

nidcpower.Session.channels[]

```
session.channels['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

instruments

nidcpower.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-DCPower

ApertureTimeUnits

class nidcpower.**ApertureTimeUnits**

SECONDS
Specifies aperture time in seconds.

POWER_LINE_CYCLES
Specifies aperture time in power line cycles (PLCs).

AutoZero

class nidcpower.**AutoZero**

OFF
Disables auto zero.

ON
Makes zero conversions for every measurement.

ONCE
Makes zero conversions following the first measurement after initiating the device. The device uses these zero conversions for the preceding measurement and future measurements until the device is reinitiated.

AutorangeApertureTimeMode

class nidcpower.**AutorangeApertureTimeMode**

AUTO
NI-DCPower optimizes the aperture time for the autorange algorithm based on the module range.

CUSTOM
The user specifies a minimum aperture time for the algorithm using the *nidcpower.Session.autorange_minimum_aperture_time* property and the corresponding *nidcpower.Session.autorange_minimum_aperture_time_units* property.

AutorangeBehavior

class nidcpower.**AutorangeBehavior**

UP_TO_LIMIT_THEN_DOWN
Go to limit range then range down as needed until measured value is within thresholds.

UP
go up one range when the upper threshold is reached.

UP_AND_DOWN
go up or down one range when the upper/lower threshold is reached.

AutorangeThresholdMode

class nidcpower.**AutorangeThresholdMode**

NORMAL
Thresholds are selected based on a balance between accuracy and hysteresis.

FAST_STEP
Optimized for faster changes in the measured signal. Thresholds are configured to be a smaller percentage of the range.

HIGH_HYSTERESIS
Optimized for noisy signals to minimize frequent and unpredictable range changes. Thresholds are configured to be a larger percentage of the range.

MEDIUM_HYSTERESIS
Optimized for noisy signals to minimize frequent and unpredictable range changes. Thresholds are configured to be a medium percentage of the range.

HOLD
Attempt to maintain the active range. Thresholds will favor the active range.

ComplianceLimitSymmetry

class nidcpower.**ComplianceLimitSymmetry**

SYMMETRIC
Compliance limits are specified symmetrically about 0.

ASYMMETRIC
Compliance limits can be specified asymmetrically with respect to 0.

DCNoiseRejection

class `nidcpower.`**`DCNoiseRejection`**

`SECOND_ORDER`
Second-order rejection of DC noise.

`NORMAL`
Normal rejection of DC noise.

Event

class `nidcpower.`**`Event`**

`SOURCE_COMPLETE`

`MEASURE_COMPLETE`

`SEQUENCE_ITERATION_COMPLETE`

`SEQUENCE_ENGINE_DONE`

`PULSE_COMPLETE`

`READY_FOR_PULSE_TRIGGER`

MeasureWhen

class `nidcpower.`**`MeasureWhen`**

`AUTOMATICALLY_AFTER_SOURCE_COMPLETE`
Acquires a measurement after each Source Complete event completes.

`ON_DEMAND`
Acquires a measurement when the *`nidcpower.Session.measure()`* method or *`nidcpower.Session.measure_multiple()`* method is called.

`ON_MEASURE_TRIGGER`
Acquires a measurement when a Measure trigger is received.

MeasurementTypes

class `nidcpower.`**`MeasurementTypes`**

`CURRENT`
The device measures current.

`VOLTAGE`
The device measures voltage.

OutputCapacitance

class nidcpower.**OutputCapacitance**

LOW
Output Capacitance is low.

HIGH
Output Capacitance is high.

OutputCutoffReason

class nidcpower.**OutputCutoffReason**

ALL
Queries any output cutoff condition; clears all output cutoff conditions.

VOLTAGE_OUTPUT_HIGH
Queries or clears cutoff conditions when the output exceeded the high cutoff limit for voltage output.

VOLTAGE_OUTPUT_LOW
Queries or clears cutoff conditions when the output fell below the low cutoff limit for voltage output.

CURRENT_MEASURE_HIGH
Queries or clears cutoff conditions when the measured current exceeded the high cutoff limit for current output.

CURRENT_MEASURE_LOW
Queries or clears cutoff conditions when the measured current fell below the low cutoff limit for current output.

VOLTAGE_CHANGE_HIGH
Queries or clears cutoff conditions when the voltage slew rate increased beyond the positive change cutoff for voltage output.

VOLTAGE_CHANGE_LOW
Queries or clears cutoff conditions when the voltage slew rate decreased beyond the negative change cutoff for voltage output.

CURRENT_CHANGE_HIGH
Queries or clears cutoff conditions when the current slew rate increased beyond the positive change cutoff for current output.

CURRENT_CHANGE_LOW
Queries or clears cutoff conditions when the current slew rate decreased beyond the negative change cutoff for current output.

OutputFunction

class nidcpower.**OutputFunction**

DC_VOLTAGE
Sets the output method to DC voltage.

DC_CURRENT
Sets the output method to DC current.

PULSE_VOLTAGE
Sets the output method to pulse voltage.

PULSE_CURRENT
Sets the output method to pulse current.

OutputStates

class nidcpower.**OutputStates**

VOLTAGE
The device maintains a constant voltage by adjusting the current

CURRENT
The device maintains a constant current by adjusting the voltage.

Polarity

class nidcpower.**Polarity**

HIGH
A high pulse occurs when the event is generated. The exported signal is low level both before and after the event is generated.

LOW
A low pulse occurs when the event is generated. The exported signal is high level both before and after the event is generated.

PowerAllocationMode

class nidcpower.**PowerAllocationMode**

DISABLED
The device attempts to source, on each active channel, the power that the present source configuration requires; NI-DCPower does not perform a sourcing power check. If the required power is greater than the maximum sourcing power, the device attempts to source the required amount and may shut down to prevent damage.

AUTOMATIC
The device attempts to source, on each active channel, the power that the present source configuration requires; NI-DCPower performs a sourcing power check. If the required power is greater than the maximum sourcing power, the device does not exceed the maximum power, and NI-DCPower returns an error.

MANUAL
The device attempts to source, on each active channel, the power you request with the *nidcpower.Session.requested_power_allocation* property; NI-DCPower performs a sourcing power check. If the requested power is either less than the required power for the present source configuration or greater than the maximum sourcing power, the device does not exceed the requested or allowed power, respectively, and NI-DCPower returns an error.

PowerSource

class nidcpower.**PowerSource**

INTERNAL
Uses the PXI chassis power source.

AUXILIARY
Uses the auxiliary power source connected to the device.

AUTOMATIC
Uses the auxiliary power source if it is available; otherwise uses the PXI chassis power source.

PowerSourceInUse

class nidcpower.**PowerSourceInUse**

INTERNAL
Uses the PXI chassis power source.

AUXILIARY
Uses the auxiliary power source connected to the device. Only the NI PXI-4110, NI PXIe-4112, NI PXIe-4113, and NI PXI-4130 support this value. This is the only supported value for the NI PXIe-4112 and NI PXIe-4113.

SelfCalibrationPersistence

class nidcpower.**SelfCalibrationPersistence**

KEEP_IN_MEMORY
Keep new self calibration values in memory only.

WRITE_TO_EEPROM
Write new self calibration values to hardware.

SendSoftwareEdgeTriggerType

class nidcpower.**SendSoftwareEdgeTriggerType**

START

SOURCE

MEASURE

SEQUENCE_ADVANCE

PULSE

SHUTDOWN

Sense

`class nidcpower.Sense`

`LOCAL`
Local sensing is selected.

`REMOTE`
Remote sensing is selected.

SourceMode

`class nidcpower.SourceMode`

`SINGLE_POINT`
The source unit applies a single source configuration.

`SEQUENCE`
The source unit applies a list of voltage or current configurations sequentially.

TransientResponse

`class nidcpower.TransientResponse`

`NORMAL`
The output responds to changes in load at a normal speed.

`FAST`
The output responds to changes in load quickly.

`SLOW`
The output responds to changes in load slowly.

`CUSTOM`
The output responds to changes in load based on specified values.

TriggerType

`class nidcpower.TriggerType`

`NONE`
No trigger is configured.

`DIGITAL_EDGE`
The data operation starts when a digital edge is detected.

`SOFTWARE_EDGE`
The data operation starts when a software trigger occurs.

Exceptions and Warnings

Error

exception `nidcpower.errors.`**`Error`**

Base exception type that all NI-DCPower exceptions derive from

DriverError

exception `nidcpower.errors.`**`DriverError`**

An error originating from the NI-DCPower driver

UnsupportedConfigurationError

exception `nidcpower.errors.`**`UnsupportedConfigurationError`**

An error due to using this module in an usupported platform.

DriverNotInstalledError

exception `nidcpower.errors.`**`DriverNotInstalledError`**

An error due to using this module without the driver runtime installed.

InvalidRepeatedCapabilityError

exception `nidcpower.errors.`**`InvalidRepeatedCapabilityError`**

An error due to an invalid character in a repeated capability

SelfTestError

exception `nidcpower.errors.`**`SelfTestError`**

An error due to a failed self-test

DriverWarning

exception `nidcpower.errors.`**`DriverWarning`**

A warning originating from the NI-DCPower driver

Examples

You can download all nidcpower examples here

nidcpower_advanced_sequence.py

Listing 1: (nidcpower_advanced_sequence.py)

```
#!/usr/bin/python

import argparse
import hightime
import nidcpower
import sys


def example(resource_name, options, voltage_max, current_max, points_per_output_
↪function, delay_in_seconds):
    timeout = hightime.timedelta(seconds=(delay_in_seconds + 1.0))

    with nidcpower.Session(resource_name=resource_name, options=options) as session:

        # Configure the session.
        session.source_mode = nidcpower.SourceMode.SEQUENCE
        session.voltage_level_autorange = True
        session.current_limit_autorange = True
        session.source_delay = hightime.timedelta(seconds=delay_in_seconds)
        properties_used = ['output_function', 'voltage_level', 'current_level']
        session.create_advanced_sequence(sequence_name='my_sequence', property_
↪names=properties_used, set_as_active_sequence=True)

        voltage_per_step = voltage_max / points_per_output_function
        for i in range(points_per_output_function):
            session.create_advanced_sequence_step(set_as_active_step=False)
            session.output_function = nidcpower.OutputFunction.DC_VOLTAGE
            session.voltage_level = voltage_per_step * i

        current_per_step = current_max / points_per_output_function
        for i in range(points_per_output_function):
            session.create_advanced_sequence_step(set_as_active_step=False)
            session.output_function = nidcpower.OutputFunction.DC_CURRENT
            session.current_level = current_per_step * i

        with session.initiate():
            session.wait_for_event(nidcpower.Event.SEQUENCE_ENGINE_DONE)
            channel_indices = '0-{0}'.format(session.channel_count - 1)
            channels = session.get_channel_names(channel_indices)
            measurement_group = [session.channels[name].fetch_multiple(points_per_
↪output_function * 2, timeout=timeout) for name in channels]

        session.delete_advanced_sequence(sequence_name='my_sequence')
        line_format = '{:<15} {:<4} {:<10} {:<10} {:<6}'
        print(line_format.format('Channel', 'Num', 'Voltage', 'Current', 'In␣
↪Compliance'))
        for i, measurements in enumerate(measurement_group):
            num = 0
            channel_name = channels[i].strip()
            for measurement in measurements:
                print(line_format.format(channel_name, num, measurement.voltage,␣
↪measurement.current, str(measurement.in_compliance)))
                num += 1


```

(continues on next page)

(continued from previous page)

```
def _main(argsv):
    parser = argparse.ArgumentParser(description='Output ramping voltage to voltage␣
↪max, then ramping current to current max.', formatter_class=argparse.
↪ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2/0, PXI1Slot3/0-1',
↪ help='Resource name of National Instruments SMUs')
    parser.add_argument('-s', '--number-steps', default=256, help='Number of steps␣
↪per output function')
    parser.add_argument('-v', '--voltage-max', default=1.0, type=float, help='Maximum␣
↪voltage (V)')
    parser.add_argument('-i', '--current-max', default=0.001, type=float, help=
↪'Maximum Current (I)')
    parser.add_argument('-d', '--delay', default=0.05, type=float, help='Source delay␣
↪(s)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.voltage_max, args.current_
↪max, args.number_steps, args.delay)


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


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:4162; BoardType:PXIe
↪', ]
    _main(cmd_line)


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '4162', 'BoardType': 'PXIe
↪', }, }
    example('PXI1Slot2/0, PXI1Slot3/1', options, 1.0, 0.001, 256, 0.05)


if __name__ == '__main__':
    main()


```

nidcpower_measure_record.py

Listing 2: (nidcpower_measure_record.py)

```
#!/usr/bin/python

import argparse
import nidcpower
import sys


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

(continues on next page)

(continued from previous page)

```
    with nidcpower.Session(resource_name=resource_name, options=options) as session:

        # Configure the session.
        session.measure_record_length = length
        session.measure_record_length_is_finite = True
        session.measure_when = nidcpower.MeasureWhen.AUTOMATICALLY_AFTER_SOURCE_
↪COMPLETE
        session.voltage_level = voltage

        session.commit()
        print('Effective measurement rate: {0} S/s'.format(session.measure_record_
↪delta_time / 1))

        print('Channel           Num  Voltage    Current    In Compliance')
        row_format = '{0:15} {1:3d}    {2:8.6f}    {3:8.6f}    {4}'
        with session.initiate():
            channel_indices = '0-{0}'.format(session.channel_count - 1)
            channels = session.get_channel_names(channel_indices)
            for i, channel_name in enumerate(channels):
                samples_acquired = 0
                while samples_acquired < length:
                    measurements = session.channels[channel_name].fetch_
↪multiple(count=session.fetch_backlog)
                    samples_acquired += len(measurements)
                    for i in range(len(measurements)):
                        print(row_format.format(channel_name, i, measurements[i].
↪voltage, measurements[i].current, measurements[i].in_compliance))


def _main(argsv):
    parser = argparse.ArgumentParser(description='Outputs the specified voltage, then␣
↪takes the specified number of voltage and current readings.', formatter_
↪class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2/0, PXI1Slot3/0-1',
↪ help='Resource names of National Instruments SMUs')
    parser.add_argument('-l', '--length', default='20', type=int, help='Measure␣
↪record length per channel')
    parser.add_argument('-v', '--voltage', default=5.0, type=float, help='Voltage␣
↪level (V)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.voltage, args.length)


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


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '4162', 'BoardType': 'PXIe
↪', }, }
    example('PXI1Slot2/0, PXI1Slot3/1', options, 5.0, 20)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:4162; BoardType:PXIe
↪', ]
```

(continues on next page)

(continued from previous page)

```python
    _main(cmd_line)


if __name__ == '__main__':
    main()


```

nidcpower_source_delay_measure.py

Listing 3: (nidcpower_source_delay_measure.py)

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

import argparse
import hightime
import nidcpower
import sys


def print_fetched_measurements(measurements):
    print('             Voltage : {:f} V'.format(measurements[0].voltage))
    print('              Current: {:f} A'.format(measurements[0].current))
    print('        In compliance: {0}'.format(measurements[0].in_compliance))


def example(resource_name, options, voltage1, voltage2, delay):
    timeout = hightime.timedelta(seconds=(delay + 1.0))

    with nidcpower.Session(resource_name=resource_name, options=options) as session:

        # Configure the session.
        session.source_mode = nidcpower.SourceMode.SINGLE_POINT
        session.output_function = nidcpower.OutputFunction.DC_VOLTAGE
        session.current_limit = .06
        session.voltage_level_range = 5.0
        session.current_limit_range = .06
        session.source_delay = hightime.timedelta(seconds=delay)
        session.measure_when = nidcpower.MeasureWhen.AUTOMATICALLY_AFTER_SOURCE_
↪COMPLETE
        session.voltage_level = voltage1

        with session.initiate():
            channel_indices = '0-{0}'.format(session.channel_count - 1)
            channels = session.get_channel_names(channel_indices)
            for channel_name in channels:
                print('Channel: {0}'.format(channel_name))
                print('---------------------------------')
                print('Voltage 1:')
                print_fetched_measurements(session.channels[channel_name].fetch_
↪multiple(count=1, timeout=timeout))
                session.voltage_level = voltage2  # on-the-fly set
                print('Voltage 2:')
                print_fetched_measurements(session.channels[channel_name].fetch_
↪multiple(count=1, timeout=timeout))
```

(continues on next page)

(continued from previous page)

```
                session.output_enabled = False
                print('')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Outputs voltage 1, waits for source delay, and then takes a measurement. Then orepeat with voltage 2.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2/0, PXI1Slot3/0-1', help='Resource name of National Instruments SMUs')
    parser.add_argument('-v1', '--voltage1', default=1.0, type=float, help='Voltage level 1 (V)')
    parser.add_argument('-v2', '--voltage2', default=2.0, type=float, help='Voltage level 2 (V)')
    parser.add_argument('-d', '--delay', default=0.05, type=float, help='Source delay (s)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.voltage1, args.voltage2, args.delay)


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


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:4162; BoardType:PXIe', ]
    _main(cmd_line)


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '4162', 'BoardType': 'PXIe', }, }
    example('PXI1Slot2/0, PXI1Slot3/1', options, 1.0, 2.0, 0.05)


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

7.2 nidigital module

7.2.1 Installation

As a prerequisite to using the nidigital module, you must install the NI-Digital Pattern Driver runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

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

```
$ python -m pip install nidigital~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install nidigital
```

7.2.2 Usage

The following is a basic example of using the **nidigital** module to open a session to a digital pattern instrument, source current, and measure both voltage and current using the PPMU on selected channels.

```
import nidigital
import time

with nidigital.Session(resource_name='PXI1Slot2') as session:

    channels = 'PXI1Slot2/0,PXI1Slot2/1'

    # Configure PPMU measurements
    session.channels[channels].ppmu_aperture_time = 0.000004
    session.channels[channels].ppmu_aperture_time_units = nidigital.
↪PPMUApertureTimeUnits.SECONDS

    session.channels[channels].ppmu_output_function = nidigital.PPMUOutputFunction.
↪CURRENT

    session.channels[channels].ppmu_current_level_range = 0.000002
    session.channels[channels].ppmu_current_level = 0.000002
    session.channels[channels].ppmu_voltage_limit_high = 3.3
    session.channels[channels].ppmu_voltage_limit_low = 0

    # Sourcing
    session.channels[channels].ppmu_source()

    # Settling time between sourcing and measuring
    time.sleep(0.01)

    # Measuring
    current_measurements = session.channels[channels].ppmu_measure(nidigital.
↪PPMUMeasurementType.CURRENT)
    voltage_measurements = session.channels[channels].ppmu_measure(nidigital.
↪PPMUMeasurementType.VOLTAGE)

    print('{:<20} {:<10} {:<10}'.format('Channel Name', 'Current', 'Voltage'))
    for channel, current, voltage in zip(channels.split(','), current_measurements,␣
↪voltage_measurements):
        print('{:<20} {:<10f} {:<10f}'.format(channel, current, voltage))

    # Disconnect all channels using programmable onboard switching
    session.channels[channels].selected_function = nidigital.SelectedFunction.
↪DISCONNECT
```

Additional examples for NI-Digital Pattern Driver are located in src/nidigital/examples/ directory.

7.2.3 API Reference

Session

class `nidigital.`**`Session`**(*self*, *resource_name*, *id_query=False*, *reset_device=False*, *options={}*)

Creates and returns a new session to the specified digital pattern instrument to use in all subsequent method calls. To place the instrument in a known startup state when creating a new session, set the reset parameter to True, which is equivalent to calling the *`nidigital.Session.reset()`* method immediately after initializing the session.

Parameters

- **`resource_name`** (*`str`*) – The specified resource name shown in Measurement & Automation Explorer (MAX) for a digital pattern instrument, for example, PXI1Slot3, where PXI1Slot3 is an instrument resource name. **resourceName** can also be a logical IVI name. This parameter accepts a comma-delimited list of strings in the form PXI1Slot2,PXI1Slot3, where `PXI1Slot2` is one instrument resource name and `PXI1Slot3` is another. When including more than one digital pattern instrument in the comma-delimited list of strings, list the instruments in the same order they appear in the pin map.

	Note You only can specify multiple instruments of the same model. For example, you can list two PXIe-6570s but not a PXIe-6570 and PXIe-6571. The instruments must be in the same chassis.

- **`id_query`** (*`bool`*) – A Boolean that verifies that the digital pattern instrument you initialize is supported by NI-Digital. NI-Digital automatically performs this query, so setting this parameter is not necessary.
- **`reset_device`** (*`bool`*) – A Boolean that specifies whether to reset a digital pattern instrument to a known state when the session is initialized. Setting the **resetDevice** value to True is equivalent to calling the *`nidigital.Session.reset()`* method immediately after initializing the session.
- **`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	{}

Methods

abort

`nidigital.Session.abort()`

Stops bursting the pattern.

abort_keep_alive

`nidigital.Session.abort_keep_alive()`

Stops the keep alive pattern if it is currently running. If a pattern burst is in progress, the method aborts the pattern burst. If you start a new pattern burst while a keep alive pattern is running, the keep alive pattern runs to the last keep alive vector, and the new pattern burst starts on the next cycle.

apply_levels_and_timing

`nidigital.Session.apply_levels_and_timing`(*levels_sheet*, *timing_sheet*, *initial_state_high_pins=None*, *initial_state_low_pins=None*, *initial_state_tristate_pins=None*)

Applies digital levels and timing values defined in previously loaded levels and timing sheets. When applying a levels sheet, only the levels specified in the sheet are affected. Any levels not specified in the sheet remain unchanged. When applying a timing sheet, all existing time sets are deleted before the new time sets are loaded.

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

Example: `my_session.sites[ ... ].apply_levels_and_timing()`

To call the method on all sites, you can call it directly on the `nidigital.Session`.

Example: `my_session.apply_levels_and_timing()`

Parameters

- **`levels_sheet`** (*str*) – Name of the levels sheet to apply. Use the name of the sheet or pass the absolute file path you use in the `nidigital.Session.load_specifications_levels_and_timing()` method. The name of the levels sheet is the file name without the directory and file extension.
- **`timing_sheet`** (*str*) – Name of the timing sheet to apply. Use the name of the sheet or pass the absolute file path that you use in the `nidigital.Session.load_specifications_levels_and_timing()` method. The name of the timing sheet is the file name without the directory and file extension.
- **`initial_state_high_pins`** (*basic sequence types or str*) – Comma-delimited list of pins, pin groups, or channels to initialize to a high state.
- **`initial_state_low_pins`** (*basic sequence types or str*) – Comma-delimited list of pins, pin groups, or channels to initialize to a low state.
- **`initial_state_tristate_pins`** (*basic sequence types or str*) – Comma-delimited list of pins, pin groups, or channels to initialize to a non-drive state (X)

apply_tdr_offsets

`nidigital.Session.`**`apply_tdr_offsets`**`(`*offsets*`)`

Applies the correction for propagation delay offsets to a digital pattern instrument. Use this method to apply TDR offsets that are stored from a past measurement or are measured by means other than the *`nidigital.Session.tdr()`* method. Also use this method to apply correction for offsets if the **applyOffsets** input of the *`nidigital.Session.tdr()`* method was set to False at the time of measurement.

Tip: This method can be called on specific channels within your *`nidigital.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[ ... ].apply_tdr_offsets()`

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

Example: `my_session.apply_tdr_offsets()`

Parameters **`offsets`** (*`basic sequence of hightime.timedelta, datetime.timedelta, or float in seconds`*) – TDR offsets to apply, in seconds. Specify an offset for each pin or channel in the repeated capabilities. If the repeated capabilities contain pin names, you must specify offsets for each site in the channel map per pin.

burst_pattern

`nidigital.Session.`**`burst_pattern`**`(`*start_label*, *select_digital_function=True*, *wait_until_done=True*, *timeout=hightime.timedelta(seconds=10.0)*`)`

Uses the start_label you specify to burst the pattern on the sites you specify. If you specify wait_until_done as True, waits for the burst to complete, and returns comparison results for each site.

Digital pins retain their state at the end of a pattern burst until the first vector of the pattern burst, a call to *`nidigital.Session.write_static()`*, or a call to *`nidigital.Session.apply_levels_and_timing()`*.

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

Example: `my_session.sites[ ... ].burst_pattern()`

To call the method on all sites, you can call it directly on the *`nidigital.Session`*.

Example: `my_session.burst_pattern()`

Parameters

- **`start_label`** (*`str`*) – Pattern name or exported pattern label from which to start bursting the pattern.

- **select_digital_function** (*bool*) – A Boolean that specifies whether to select the digital method for the pins in the pattern prior to bursting.
- **wait_until_done** (*bool*) – A Boolean that indicates whether to wait until the bursting is complete.
- **timeout** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Maximum time (in seconds) allowed for this method to complete. If this method does not complete within this time interval, this method returns an error.

Return type { int: bool, int: bool, .. }

Returns Dictionary where each key is a site number and value is pass/fail, if wait_until_done is specified as True. Else, None.

clock_generator_abort

`nidigital.Session.`**`clock_generator_abort`**`()`

Stops clock generation on the specified channel(s) or pin(s) and pin group(s).

Tip: This method can be called on specific channels within your `nidigital.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[ ... ].clock_generator_abort()`

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

Example: `my_session.clock_generator_abort()`

clock_generator_generate_clock

`nidigital.Session.`**`clock_generator_generate_clock`**`(`*frequency*, *select_digital_function=True*`)`

Configures clock generator frequency and initiates clock generation on the specified channel(s) or pin(s) and pin group(s).

Tip: This method can be called on specific channels within your `nidigital.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[ ... ].clock_generator_generate_clock()`

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

Example: `my_session.clock_generator_generate_clock()`

Parameters

- **frequency** (*float*) – The frequency of the clock generation, in Hz.
- **select_digital_function** (*bool*) – A Boolean that specifies whether to select the digital method for the pins specified prior to starting clock generation.

close

`nidigital.Session.close()`

Closes the specified instrument session to a digital pattern instrument, aborts pattern execution, and unloads pattern memory. The channels on a digital pattern instrument remain in their current state.

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

commit

`nidigital.Session.commit()`

Applies all previously configured pin levels, termination modes, clocks, triggers, and pattern timing to a digital pattern instrument. If you do not call the `nidigital.Session.commit()` method, then the initiate method or the `nidigital.Session.burst_pattern()` method will implicitly call this method for you. Calling this method moves the session from the Uncommitted state to the Committed state.

configure_active_load_levels

`nidigital.Session.configure_active_load_levels(iol, ioh, vcom)`

Configures I_{OL}, I_{OH}, and V_{COM} levels for the active load on the pins you specify. The DUT sources or sinks current based on the level values. To enable active load, set the termination mode to `ACTIVE_LOAD`. To disable active load, set the termination mode of the instrument to `HIGH_Z` or `VTERM`.

Tip: This method can be called on specific channels within your `nidigital.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_active_load_levels()`

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

Example: `my_session.configure_active_load_levels()`

Parameters

- **iol** (*float*) – Maximum current that the DUT sinks while outputting a voltage below V_{COM}.
- **ioh** (*float*) – Maximum current that the DUT sources while outputting a voltage above V_{COM}.
- **vcom** (*float*) – Commutating voltage level at which the active load circuit switches between sourcing current and sinking current.

configure_pattern_burst_sites

`nidigital.Session.configure_pattern_burst_sites()`

Configures which sites burst the pattern on the next call to the initiate method. The pattern burst sites can also be modified through the repeated capabilities for the `nidigital.Session.burst_pattern()` method. If a site has been disabled through the `nidigital.Session.disable_sites()` method, the site does not burst a pattern even if included in the pattern burst sites.

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

Example: `my_session.sites[ ... ].configure_pattern_burst_sites()`

To call the method on all sites, you can call it directly on the `nidigital.Session`.

Example: `my_session.configure_pattern_burst_sites()`

configure_time_set_compare_edges_strobe

nidigital.Session.**configure_time_set_compare_edges_strobe**(*time_set_name*, *strobe_edge*)

Configures the strobe edge time for the specified pins. Use this method to modify time set values after applying a timing sheet with the `nidigital.Session.apply_levels_and_timing()` method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to `nidigital.Session.apply_levels_and_timing()`; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_compare_edges_strobe()`

To call the method on all pins, you can call it directly on the `nidigital.Session`.

Example: `my_session.configure_time_set_compare_edges_strobe()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **strobe_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Time when the comparison happens within a vector period.

configure_time_set_compare_edges_strobe2x

nidigital.Session.**configure_time_set_compare_edges_strobe2x**(*time_set_name*, *strobe_edge*, *strobe2_edge*)

Configures the compare strobes for the specified pins in the time set, including the 2x strobe. Use this method to modify time set values after applying a timing sheet with the `nidigital.Session.apply_levels_and_timing()` method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_compare_edges_strobe2x()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_compare_edges_strobe2x()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **strobe_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Time when the comparison happens within a vector period.
- **strobe2_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Time when the comparison happens for the second DUT cycle within a vector period.

configure_time_set_drive_edges

`nidigital.Session.`**`configure_time_set_drive_edges`**(*time_set_name*, *format*, *drive_on_edge*, *drive_data_edge*, *drive_return_edge*, *drive_off_edge*)

Configures the drive format and drive edge placement for the specified pins. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_drive_edges()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_drive_edges()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **format** (*nidigital.DriveFormat*) – Drive format of the time set.
  - *NR*: Non-return.

  - *RL*: Return to low.
  - *RH*: Return to high.
  - *SBC*: Surround by complement.

- **drive_on_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period for turning on the pin driver.This option applies only when the prior vector left the pin in a non-drive pin state (L, H, X, V, M, E). For the SBC format, this option specifies the delay from the beginning of the vector period at which the complement of the pattern value is driven.
- **drive_data_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pattern data is driven to the pattern value.The ending state from the previous vector persists until this point.
- **drive_return_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pin changes from the pattern data to the return value, as specified in the format.
- **drive_off_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period to turn off the pin driver when the next vector period uses a non-drive symbol (L, H, X, V, M, E).

configure_time_set_drive_edges2x

nidigital.Session.**configure_time_set_drive_edges2x**(*time_set_name*, *format*, *drive_on_edge*, *drive_data_edge*, *drive_return_edge*, *drive_off_edge*, *drive_data2_edge*, *drive_return2_edge*)

Configures the drive edges of the pins in the time set, including 2x edges. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_drive_edges2x()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_drive_edges2x()`

Parameters

- **time_set_name** (*str*) – The specified time set name.

- **format** (*nidigital.DriveFormat*) – Drive format of the time set.
  - *NR*: Non-return.
  - *RL*: Return to low.
  - *RH*: Return to high.
  - *SBC*: Surround by complement.
- **drive_on_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period for turning on the pin driver.This option applies only when the prior vector left the pin in a non-drive pin state (L, H, X, V, M, E). For the SBC format, this option specifies the delay from the beginning of the vector period at which the complement of the pattern value is driven.
- **drive_data_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pattern data is driven to the pattern value.The ending state from the previous vector persists until this point.
- **drive_return_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pin changes from the pattern data to the return value, as specified in the format.
- **drive_off_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period to turn off the pin driver when the next vector period uses a non-drive symbol (L, H, X, V, M, E).
- **drive_data2_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pattern data in the second DUT cycle is driven to the pattern value.
- **drive_return2_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pin changes from the pattern data in the second DUT cycle to the return value, as specified in the format.

configure_time_set_drive_format

nidigital.Session.**configure_time_set_drive_format**(*time_set_name*, *drive_format*)

Configures the drive format for the pins specified in the **pinList**. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_drive_format()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_drive_format()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **drive_format** (*nidigital.DriveFormat*) – Drive format of the time set.
  - *NR*: Non-return.
  - *RL*: Return to low.
  - *RH*: Return to high.
  - *SBC*: Surround by complement.

configure_time_set_edge

nidigital.Session.**configure_time_set_edge**(*time_set_name*, *edge*, *time*)

Configures the edge placement for the pins specified in the pin list. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_edge()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_edge()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **edge** (*nidigital.TimeSetEdgeType*) – Name of the edge.
  - *DRIVE_ON*
  - *DRIVE_DATA*
  - *DRIVE_RETURN*
  - *DRIVE_OFF*
  - *COMPARE_STROBE*
  - *DRIVE_DATA2*
  - *DRIVE_RETURN2*
  - *COMPARE_STROBE2*

- **time** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The time from the beginning of the vector period in which to place the edge.

configure_time_set_edge_multiplier

nidigital.Session.**configure_time_set_edge_multiplier**(*time_set_name*, *edge_multiplier*)

Configures the edge multiplier of the pins in the time set. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_edge_multiplier()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_edge_multiplier()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **edge_multiplier** (*int*) – The specified edge multiplier for the pins in the pin list.

configure_time_set_period

nidigital.Session.**configure_time_set_period**(*time_set_name*, *period*)

Configures the period of a time set. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **period** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Period for this time set, in seconds.

configure_voltage_levels

nidigital.Session.**configure_voltage_levels**(*vil*, *vih*, *vol*, *voh*, *vterm*)

Configures voltage levels for the pins you specify.

Tip: This method can be called on specific channels within your `nidigital.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_voltage_levels()`

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

Example: `my_session.configure_voltage_levels()`

Parameters

- **`vil`** (*`float`*) – Voltage that the instrument will apply to the input of the DUT when the pin driver drives a logic low (0).
- **`vih`** (*`float`*) – Voltage that the instrument will apply to the input of the DUT when the test instrument drives a logic high (1).
- **`vol`** (*`float`*) – Output voltage below which the comparator on the pin driver interprets a logic low (L).
- **`voh`** (*`float`*) – Output voltage above which the comparator on the pin driver interprets a logic high (H).
- **`vterm`** (*`float`*) – Termination voltage the instrument applies during non-drive cycles when the termination mode is set to V_{term}. The instrument applies the termination voltage through a 50 ohm parallel termination resistance.

create_capture_waveform_from_file_digicapture

`nidigital.Session.`**`create_capture_waveform_from_file_digicapture`**`(`*waveform_name*, *waveform_file_path*`)`

Creates a capture waveform with the configuration information from a Digicapture file generated by the Digital Pattern Editor.

Parameters

- **`waveform_name`** (*`str`*) – Waveform name you want to use. You must specify waveform_name if the file contains multiple waveforms. Use the waveform_name with the capture_start opcode in your pattern.
- **`waveform_file_path`** (*`str`*) – Absolute file path to the capture waveform file (.digicapture) you want to load.

create_capture_waveform_parallel

`nidigital.Session.`**`create_capture_waveform_parallel`**`(`*waveform_name*`)`

Sets the capture waveform settings for parallel acquisition. Settings apply across all sites if multiple sites are configured in the pin map. You cannot reconfigure settings after waveforms are created.

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

Example: `my_session.pins[ ... ].create_capture_waveform_parallel()`

To call the method on all pins, you can call it directly on the `nidigital.Session`.

Example: `my_session.create_capture_waveform_parallel()`

Parameters **`waveform_name`** (*`str`*) – Waveform name you want to use. Use the waveform_name with the capture_start opcode in your pattern.

create_capture_waveform_serial

`nidigital.Session.`**`create_capture_waveform_serial`**`(`*waveform_name*, *sample_width*, *bit_order*`)`

Sets the capture waveform settings for serial acquisition. Settings apply across all sites if multiple sites are configured in the pin map. You cannot reconfigure settings after waveforms are created.

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

Example: `my_session.pins[ ... ].create_capture_waveform_serial()`

To call the method on all pins, you can call it directly on the `nidigital.Session`.

Example: `my_session.create_capture_waveform_serial()`

Parameters

- **`waveform_name`** (*`str`*) – Waveform name you want to use. Use the waveform_name with the capture_start opcode in your pattern.
- **`sample_width`** (*`int`*) – Width in bits of each serial sample. Valid values are between 1 and 32.
- **`bit_order`** (*`nidigital.BitOrder`*) – Order in which to shift the bits.
  - *`MSB`*: Specifies the bit order by most significant bit first.
  - *`LSB`*: Specifies the bit order by least significant bit first.

create_source_waveform_from_file_tdms

`nidigital.Session.`**`create_source_waveform_from_file_tdms`**`(`*waveform_name*, *waveform_file_path*, *write_waveform_data=True*`)`

Creates a source waveform with configuration information from a TDMS file generated by the Digital Pattern Editor. It also optionally writes waveform data from the file.

Parameters

- **`waveform_name`** (*`str`*) – The waveform name you want to use from the file. You must specify waveform_name if the file contains multiple waveforms. Use the waveform_name with the source_start opcode in your pattern.
- **`waveform_file_path`** (*`str`*) – Absolute file path to the load source waveform file (.tdms).

- **write_waveform_data** (*bool*) – A Boolean that writes waveform data to source memory if True and the waveform data is in the file.

create_source_waveform_parallel

`nidigital.Session.`**`create_source_waveform_parallel`**(*waveform_name*, *data_mapping*)

Sets the source waveform settings required for parallel sourcing. Settings apply across all sites if multiple sites are configured in the pin map. You cannot reconfigure settings after waveforms are created.

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

Example: `my_session.pins[ ... ].create_source_waveform_parallel()`

To call the method on all pins, you can call it directly on the `nidigital.Session`.

Example: `my_session.create_source_waveform_parallel()`

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **data_mapping** (`nidigital.SourceDataMapping`) – Parameter that specifies how to map data on multiple sites.
  - *BROADCAST*: Broadcasts the waveform you specify to all sites.
  - *SITE_UNIQUE*: Sources unique waveform data to each site.

create_source_waveform_serial

`nidigital.Session.`**`create_source_waveform_serial`**(*waveform_name*, *data_mapping*, *sample_width*, *bit_order*)

Sets the source waveform settings required for serial sourcing. Settings apply across all sites if multiple sites are configured in the pin map. You cannot reconfigure settings after waveforms are created.

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

Example: `my_session.pins[ ... ].create_source_waveform_serial()`

To call the method on all pins, you can call it directly on the `nidigital.Session`.

Example: `my_session.create_source_waveform_serial()`

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **data_mapping** (*nidigital.SourceDataMapping*) – Parameter that specifies how to map data on multiple sites.
  - *BROADCAST*: Broadcasts the waveform you specify to all sites.
  - *SITE_UNIQUE*: Sources unique waveform data to each site.
- **sample_width** (*int*) – Width in bits of each serial sample. Valid values are between 1 and 32.
- **bit_order** (*nidigital.BitOrder*) – Order in which to shift the bits.
  - *MSB*: Specifies the bit order by most significant bit first.
  - *LSB*: Specifies the bit order by least significant bit first.

create_time_set

nidigital.Session.**create_time_set**(*name*)

Creates a time set with the name that you specify. Use this method when you want to create time sets programmatically rather than with a timing sheet.

Parameters **name** (*str*) – The specified name of the new time set.

delete_all_time_sets

nidigital.Session.**delete_all_time_sets**()

Deletes all time sets from instrument memory.

disable_sites

nidigital.Session.**disable_sites**()

Disables specified sites. Disabled sites are not included in pattern bursts initiated by the initiate method or the *nidigital.Session.burst_pattern()* method, even if the site is specified in the list of pattern burst sites in *nidigital.Session.configure_pattern_burst_sites()* method or in the repeated capabilities for the *nidigital.Session.burst_pattern()* method. Additionally, if you specify a list of pin or pin group names in repeated capabilities in any NI-Digital method, digital pattern instrument channels mapped to disabled sites are not affected by the method. The methods that return per-pin data, such as the *nidigital.Session.ppmu_measure()* method, do not return data for channels mapped to disabled sites. The digital pattern instrument channels mapped to the sites specified are left in their current state. NI TestStand Semiconductor Module requires all sites to always be enabled, and manages the set of active sites without disabling the sites in the digital instrument session. Do not use this method with the Semiconductor Module.

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

Example: `my_session.sites[ ... ].disable_sites()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.disable_sites()`

enable_sites

`nidigital.Session.`**`enable_sites`**`()`

Enables the sites you specify. All sites are enabled by default.

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

Example: `my_session.sites[ ... ].enable_sites()`

To call the method on all sites, you can call it directly on the `nidigital.Session`.

Example: `my_session.enable_sites()`

fetch_capture_waveform

`nidigital.Session.`**`fetch_capture_waveform`**`(`*waveform_name*, *samples_to_read*, *timeout=hightime.timedelta(seconds=10.0)*`)`

Returns dictionary where each key is a site number and value is a collection of digital states representing capture waveform data

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

Example: `my_session.sites[ ... ].fetch_capture_waveform()`

To call the method on all sites, you can call it directly on the `nidigital.Session`.

Example: `my_session.fetch_capture_waveform()`

Parameters

- **`waveform_name`** (*`str`*) – Waveform name you create with the create capture waveform method. Use the waveform_name parameter with capture_start opcode in your pattern.
- **`samples_to_read`** (*`int`*) – Number of samples to fetch.
- **`timeout`** (*`hightime.timedelta, datetime.timedelta, or float in seconds`*) – Maximum time (in seconds) allowed for this method to complete. If this method does not complete within this time interval, this method returns an error.

Return type { int: memoryview of array.array of unsigned int, int: memoryview of array.array of unsigned int, .. }

Returns Dictionary where each key is a site number and value is a collection of digital states representing capture waveform data

fetch_history_ram_cycle_information

nidigital.Session.**fetch_history_ram_cycle_information**(*position*, *samples_to_read*)

Returns the pattern information acquired for the specified cycles.

If the pattern is using the edge multiplier feature, cycle numbers represent tester cycles, each of which may consist of multiple DUT cycles. When using pins with mixed edge multipliers, pins may return *PIN_STATE_NOT_ACQUIRED* for DUT cycles where those pins do not have edges defined.

Site number on which to retrieve pattern information must be specified via sites repeated capability. The method returns an error if more than one site is specified.

Pins for which to retrieve pattern information must be specified via pins repeated capability. If pins are not specified, pin list from the pattern containing the start label is used. Call *nidigital.Session.get_pattern_pin_names()* with the start label to retrieve the pins associated with the pattern burst:

```
session.sites[0].pins['PinA', 'PinB'].fetch_history_ram_cycle_
↪information(0, -1)
```

Note: Before bursting a pattern, you must configure the History RAM trigger and specify which cycles to acquire.

nidigital.Session.history_ram_trigger_type should be used to specify the trigger condition on which History RAM starts acquiring pattern information.

If History RAM trigger is configured as *CYCLE_NUMBER*, *nidigital.Session.cycle_number_history_ram_trigger_cycle_number* should be used to specify the cycle number on which History RAM starts acquiring pattern information.

If History RAM trigger is configured as *PATTERN_LABEL*, *nidigital.Session.pattern_label_history_ram_trigger_label* should be used to specify the pattern label from which to start acquiring pattern information. *nidigital.Session.pattern_label_history_ram_trigger_vector_offset* should be used to specify the number of vectors following the specified pattern label from which to start acquiring pattern information. *nidigital.Session.pattern_label_history_ram_trigger_cycle_offset* should be used to specify the number of cycles following the specified pattern label and vector offset from which to start acquiring pattern information.

For all History RAM trigger conditions, *nidigital.Session.history_ram_pretrigger_samples* should be used to specify the number of samples to acquire before the trigger conditions are met. If you configure History RAM to only acquire failed cycles, you must set *nidigital.Session.history_ram_pretrigger_samples* to 0.

nidigital.Session.history_ram_cycles_to_acquire should be used to specify which cycles History RAM acquires after the trigger conditions are met.

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

Example: `my_session.pins[ ... ].fetch_history_ram_cycle_information()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.fetch_history_ram_cycle_information()`

Parameters

- **`position`** (*int*) – Sample index from which to start fetching pattern information.
- **`samples_to_read`** (*int*) – Number of samples to fetch. A value of -1 specifies to fetch all available samples.

Return type list of HistoryRAMCycleInformation

Returns

Returns a list of class instances with the following information about each pattern cycle:

- **pattern_name** (str) Name of the pattern for the acquired cycle.
- **time_set_name** (str) Time set for the acquired cycle.
- **vector_number** (int) Vector number within the pattern for the acquired cycle. Vector numbers start at 0 from the beginning of the pattern.
- **cycle_number** (int) Cycle number acquired by this History RAM sample. Cycle numbers start at 0 from the beginning of the pattern burst.
- **scan_cycle_number** (int) Scan cycle number acquired by this History RAM sample. Scan cycle numbers start at 0 from the first cycle of the scan vector. Scan cycle numbers are -1 for cycles that do not have a scan opcode.
- **expected_pin_states** (list of list of enums.PinState) Pin states as expected by the loaded pattern in the order specified in the pin list. Pins without defined edges in the specified DUT cycle will have a value of *`PIN_STATE_NOT_ACQUIRED`*. Length of the outer list will be equal to the value of edge multiplier for the given vector. Length of the inner list will be equal to the number of pins requested.
- **actual_pin_states** (list of list of enums.PinState) Pin states acquired by History RAM in the order specified in the pin list. Pins without defined edges in the specified DUT cycle will have a value of *`PIN_STATE_NOT_ACQUIRED`*. Length of the outer list will be equal to the value of edge multiplier for the given vector. Length of the inner list will be equal to the number of pins requested.
- **per_pin_pass_fail** (list of list of bool) Pass fail information for pins in the order specified in the pin list. Pins without defined edges in the specified DUT cycle will have a value of pass (True). Length of the outer list will be equal to the value of edge multiplier for the given vector. Length of the inner list will be equal to the number of pins requested.

frequency_counter_measure_frequency

`nidigital.Session.`**`frequency_counter_measure_frequency`**`()`

Measures the frequency on the specified channel(s) over the specified measurement time. All channels in the repeated capabilities should have the same measurement time.

Tip: This method can be called on specific channels within your *`nidigital.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[ ... ].frequency_counter_measure_frequency()`

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

Example: `my_session.frequency_counter_measure_frequency()`

Return type list of float

Returns The returned frequency counter measurement, in Hz.This method returns -1 if the measurement is invalid for the channel.

get_channel_names

`nidigital.Session.`**`get_channel_names`**`(`*indices*`)`

Returns a list of channel names for given channel indices.

Parameters **`indices`** (*`basic sequence types or 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_fail_count

`nidigital.Session.`**`get_fail_count`**`()`

Returns the comparison fail count for pins in the repeated capabilities.

Tip: This method can be called on specific channels within your *`nidigital.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_fail_count()`

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

Example: `my_session.get_fail_count()`

Return type list of int

Returns Number of failures in an array. If a site is disabled or not enabled for burst, the method does not return data for that site. You can also use the *`nidigital.Session.get_pin_results_pin_information()`* method to obtain a sorted list of returned sites and channels.

get_history_ram_sample_count

nidigital.Session.**get_history_ram_sample_count**()

Returns the number of samples History RAM acquired on the last pattern burst.

Note: Before bursting a pattern, you must configure the History RAM trigger and specify which cycles to acquire.

nidigital.Session.history_ram_trigger_type should be used to specify the trigger condition on which History RAM starts acquiring pattern information.

If History RAM trigger is configured as *CYCLE_NUMBER*, *nidigital.Session.cycle_number_history_ram_trigger_cycle_number* should be used to specify the cycle number on which History RAM starts acquiring pattern information.

If History RAM trigger is configured as *PATTERN_LABEL*, *nidigital.Session.pattern_label_history_ram_trigger_label* should be used to specify the pattern label from which to start acquiring pattern information. *nidigital.Session.pattern_label_history_ram_trigger_vector_offset* should be used to specify the number of vectors following the specified pattern label from which to start acquiring pattern information. *nidigital.Session.pattern_label_history_ram_trigger_cycle_offset* should be used to specify the number of cycles following the specified pattern label and vector offset from which to start acquiring pattern information.

For all History RAM trigger conditions, *nidigital.Session.history_ram_pretrigger_samples* should be used to specify the number of samples to acquire before the trigger conditions are met. If you configure History RAM to only acquire failed cycles, you must set *nidigital.Session.history_ram_pretrigger_samples* to 0.

nidigital.Session.history_ram_cycles_to_acquire should be used to specify which cycles History RAM acquires after the trigger conditions are met.

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

Example: `my_session.sites[ ... ].get_history_ram_sample_count()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.get_history_ram_sample_count()`

Return type int

Returns The returned number of samples that History RAM acquired.

get_pattern_pin_names

nidigital.Session.**get_pattern_pin_names**(*start_label*)

Returns the pattern pin list.

Parameters **start_label** (*str*) – Pattern name or exported pattern label from which to get the pin names that the pattern references.

Return type list of str

Returns List of pins referenced by the pattern with the **startLabel**.

get_pin_results_pin_information

nidigital.Session.**get_pin_results_pin_information**()

Returns the pin names, site numbers, and channel names that correspond to per-pin data read from the digital pattern instrument. The method returns pin information in the same order as values read using the *nidigital.Session.read_static()* method, *nidigital.Session.ppmu_measure()* method, and *nidigital.Session.get_fail_count()* method. Use this method to match values the previously listed methods return with pins, sites, and instrument channels.

Tip: This method can be called on specific channels within your *nidigital.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_pin_results_pin_information()`

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

Example: `my_session.get_pin_results_pin_information()`

Return type list of PinInfo

Returns

List of named tuples with fields:

- **pin_name** (str)
- **site_number** (int)
- **channel_name** (str)

get_site_pass_fail

nidigital.Session.**get_site_pass_fail**()

Returns dictionary where each key is a site number and value is pass/fail

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

Example: `my_session.sites[ ... ].get_site_pass_fail()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.get_site_pass_fail()`

Return type { int: bool, int: bool, .. }

Returns Dictionary where each key is a site number and value is pass/fail

get_time_set_drive_format

`nidigital.Session.`**`get_time_set_drive_format`**(*time_set_name*)
Returns the drive format of a pin in the specified time set.

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

Example: `my_session.pins[ ... ].get_time_set_drive_format()`

To call the method on all pins, you can call it directly on the *`nidigital.Session`*.

Example: `my_session.get_time_set_drive_format()`

Parameters **`time_set_name`** (*`str`*) – The specified time set name.

Return type *`nidigital.DriveFormat`*

Returns Returned drive format of the time set for the specified pin.

get_time_set_edge

`nidigital.Session.`**`get_time_set_edge`**(*time_set_name*, *edge*)
Returns the edge time of a pin in the specified time set.

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

Example: `my_session.pins[ ... ].get_time_set_edge()`

To call the method on all pins, you can call it directly on the *`nidigital.Session`*.

Example: `my_session.get_time_set_edge()`

Parameters

- **`time_set_name`** (*`str`*) – The specified time set name.
- **`edge`** (*`nidigital.TimeSetEdgeType`*) – Name of the edge.
  - *`DRIVE_ON`*
  - *`DRIVE_DATA`*
  - *`DRIVE_RETURN`*
  - *`DRIVE_OFF`*
  - *`COMPARE_STROBE`*
  - *`DRIVE_DATA2`*
  - *`DRIVE_RETURN2`*
  - *`COMPARE_STROBE2`*

Return type hightime.timedelta

Returns Time from the beginning of the vector period in which to place the edge.

get_time_set_edge_multiplier

nidigital.Session.**get_time_set_edge_multiplier**(*time_set_name*)
Returns the edge multiplier of the specified time set.

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

Example: `my_session.pins[ ... ].get_time_set_edge_multiplier()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.get_time_set_edge_multiplier()`

Parameters **time_set_name** (*str*) – The specified time set name.

Return type int

Returns Returned edge multiplier of the time set for the specified pin.

get_time_set_period

nidigital.Session.**get_time_set_period**(*time_set_name*)
Returns the period of the specified time set.

Parameters **time_set_name** (*str*) – The specified time set name.

Return type hightime.timedelta

Returns Returned period, in seconds, that the edge is configured to.

initiate

nidigital.Session.**initiate**()
Starts bursting the pattern configured by *nidigital.Session.start_label*, causing the NI-Digital session to be committed. To stop the pattern burst, call *nidigital.Session.abort()*. If keep alive pattern is bursting when *nidigital.Session.abort()* is called or upon exiting the context manager, keep alive pattern will not be stopped. To stop the keep alive pattern, call *nidigital.Session.abort_keep_alive()*.

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

---

is_done

nidigital.Session.**is_done**()
Checks the hardware to determine if the pattern burst has completed or if any errors have occurred.

Return type bool

Returns A Boolean that indicates whether the pattern burst completed.

is_site_enabled

`nidigital.Session.`**`is_site_enabled`**`()`

Checks if a specified site is enabled.

Note: The method returns an error if more than one site is specified.

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

Example: `my_session.sites[ ... ].is_site_enabled()`

To call the method on all sites, you can call it directly on the `nidigital.Session`.

Example: `my_session.is_site_enabled()`

Return type bool

Returns Boolean value that returns whether the site is enabled or disabled.

load_pattern

`nidigital.Session.`**`load_pattern`**`(`*file_path*`)`

Loads the specified pattern file.

Parameters **`file_path`** (*`str`*) – Absolute file path of the binary .digipat pattern file to load. Specify the pattern to burst using `nidigital.Session.start_label` or the start_label parameter of the `nidigital.Session.burst_pattern()` method.

load_pin_map

`nidigital.Session.`**`load_pin_map`**`(`*file_path*`)`

Loads a pin map file. You can load only a single pin and channel map file during an NI-Digital Pattern Driver session. To switch pin maps, create a new session or call the `nidigital.Session.reset()` method.

Parameters **`file_path`** (*`str`*) – Absolute file path to a pin map file created with the Digital Pattern Editor or the NI TestStand Semiconductor Module.

load_specifications_levels_and_timing

nidigital.Session.**load_specifications_levels_and_timing**(*specifications_file_paths=None*, *levels_file_paths=None*, *timing_file_paths=None*)

Loads settings in specifications, levels, and timing sheets. These settings are not applied to the digital pattern instrument until *nidigital.Session.apply_levels_and_timing()* is called.

If the levels and timing sheets contains formulas, they are evaluated at load time. If the formulas refer to variables, the specifications sheets that define those variables must be loaded either first, or at the same time as the levels and timing sheets.

Parameters

- **specifications_file_paths** (*str or basic sequence of str*) – Absolute file path of one or more specifications files.
- **levels_file_paths** (*str or basic sequence of str*) – Absolute file path of one or more levels sheet files.
- **timing_file_paths** (*str or basic sequence of str*) – Absolute file path of one or more timing sheet files.

lock

nidigital.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 *nidigital.Session.lock()* method.
- A call to NI-Digital Pattern Driver locked the session.
- After a call to the *nidigital.Session.lock()* method returns successfully, no other threads can access the device session until you call the *nidigital.Session.unlock()* method or exit out of the with block when using lock context manager.
- Use the *nidigital.Session.lock()* method and the *nidigital.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 *nidigital.Session.lock()* method within the same thread. To completely unlock the session, you must balance each call to the *nidigital.Session.lock()* method with a call to the *nidigital.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

```python
with nidigital.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, `nidigital.Session.lock()` acts as a context manager and unlock will be called when the *with* block is exited

ppmu_measure

`nidigital.Session.`**`ppmu_measure`**`(`*measurement_type*`)`

Instructs the PPMU to measure voltage or current. This method can be called to take a voltage measurement even if the pin method is not set to PPMU.

Tip: This method can be called on specific channels within your `nidigital.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[ ... ].ppmu_measure()`

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

Example: `my_session.ppmu_measure()`

Parameters **`measurement_type`** (*`nidigital.PPMUMeasurementType`*) – Parameter that specifies whether the PPMU measures voltage or current from the DUT.

- *`CURRENT`*: The PPMU measures current from the DUT.
- *`VOLTAGE`*: The PPMU measures voltage from the DUT.

Return type list of float

Returns The returned array of measurements in the order you specify in the repeated capabilities. If a site is disabled, the method does not return data for that site. You can also use the `nidigital.Session.get_pin_results_pin_information()` method to obtain a sorted list of returned sites and channels.

ppmu_source

`nidigital.Session.`**`ppmu_source`**`()`

Starts sourcing voltage or current from the PPMU. This method automatically selects the PPMU method. Changes to PPMU source settings do not take effect until you call this method. If you modify source settings after you call this method, you must call this method again for changes in the configuration to take effect.

Tip: This method can be called on specific channels within your `nidigital.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[ ... ].ppmu_source()`

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

Example: `my_session.ppmu_source()`

---

read_sequencer_flag

nidigital.Session.**read_sequencer_flag**(*flag*)

Reads the state of a pattern sequencer flag. Use pattern sequencer flags to coordinate execution between the pattern sequencer and a runtime test program.

Parameters **flag** (*nidigital.SequencerFlag*) – The pattern sequencer flag you want to read.

- *FLAG0* ("seqflag0"): Reads pattern sequencer flag 0.
- *FLAG1* ("seqflag1"): Reads pattern sequencer flag 1.
- *FLAG2* ("seqflag2"): Reads pattern sequencer flag 2.
- *FLAG3* ("seqflag3"): Reads pattern sequencer flag 3.

Return type bool

Returns A Boolean that indicates the state of the pattern sequencer flag you specify.

read_sequencer_register

nidigital.Session.**read_sequencer_register**(*reg*)

Reads the value of a pattern sequencer register. Use pattern sequencer registers to pass numeric values between the pattern sequencer and a runtime test program. For example, you can use this method to read a register modified by the write_reg opcode during a pattern burst.

Parameters **reg** (*nidigital.SequencerRegister*) – The sequencer register to read from.

- *REGISTER0* ("reg0"): Reads sequencer register 0.
- *REGISTER1* ("reg1"): Reads sequencer register 1.
- *REGISTER2* ("reg2"): Reads sequencer register 2.
- *REGISTER3* ("reg3"): Reads sequencer register 3.
- *REGISTER4* ("reg4"): Reads sequencer register 4.
- *REGISTER5* ("reg5"): Reads sequencer register 5.
- *REGISTER6* ("reg6"): Reads sequencer register 6.
- *REGISTER7* ("reg7"): Reads sequencer register 7.
- *REGISTER8* ("reg8"): Reads sequencer register 8.
- *REGISTER9* ("reg9"): Reads sequencer register 9.
- *REGISTER10* ("reg10"): Reads sequencer register 10.
- *REGISTER11* ("reg11"): Reads sequencer register 11.
- *REGISTER12* ("reg12"): Reads sequencer register 12.
- *REGISTER13* ("reg13"): Reads sequencer register 13.
- *REGISTER14* ("reg14"): Reads sequencer register 14.
- *REGISTER15* ("reg15"): Reads sequencer register 15.

Return type int

Returns Value read from the sequencer register.

read_static

`nidigital.Session.`**`read_static`**`()`

Reads the current state of comparators for pins you specify in the repeated capabilities. If there are uncommitted changes to levels or the termination mode, this method commits the changes to the pins.

Tip: This method can be called on specific channels within your *`nidigital.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_static()`

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

Example: `my_session.read_static()`

Return type list of *`nidigital.PinState`*

Returns

The returned array of pin states read from the channels in the repeated capabilities. Data is returned in the order you specify in the repeated capabilities. If a site is disabled, the method does not return data for that site. You can also use the *`nidigital.Session.get_pin_results_pin_information()`* method to obtain a sorted list of returned sites and channels.

- *`L`*: The comparators read a logic low pin state.
- *`H`*: The comparators read a logic high pin state.
- *`M`*: The comparators read a midband pin state.
- *`V`*: The comparators read a value that is above VOH and below VOL, which can occur when you set VOL higher than VOH.

reset

`nidigital.Session.`**`reset`**`()`

Returns a digital pattern instrument to a known state. This method performs the following actions:

- Aborts pattern execution.
- Clears pin maps, time sets, source and capture waveforms, and patterns.
- Resets all properties to default values, including the *`nidigital.Session.selected_function`* property that is set to *`DISCONNECT`*, causing the I/O switches to open.
- Stops exporting all external signals and events.

reset_device

`nidigital.Session.`**`reset_device`**`()`

Returns a digital pattern instrument to a known state. This method performs the following actions:

- Aborts pattern execution.

- Clears pin maps, time sets, source and capture waveforms, and patterns.
- Resets all properties to default values, including the `nidigital.Session.selected_function` property that is set to *DISCONNECT*, causing the I/O switches to open.
- Stops export of all external signals and events.
- Clears over-temperature and over-power conditions.

self_calibrate

`nidigital.Session.`**`self_calibrate`**`()`

Performs self-calibration on a digital pattern instrument.

self_test

`nidigital.Session.`**`self_test`**`()`

Returns self test results from a digital pattern instrument. This test requires several minutes to execute.

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	Self test passed.
1	Self test failed.

send_software_edge_trigger

`nidigital.Session.`**`send_software_edge_trigger`**`(`*trigger*, *trigger_identifier*`)`

Forces a particular edge-based trigger to occur regardless of how the specified trigger is configured. You can use this method as a software override.

Parameters

- **trigger** (`nidigital.SoftwareTrigger`) – Trigger specifies the trigger you want to override.

Defined Values	
START	Overrides the Start trigger. You must specify an empty string in the trigger_identifier parameter.
CONDITIONAL_JUMP	Specifies to route a conditional jump trigger. You must specify a conditional jump trigger in the trigger_identifier parameter.

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.

- **trigger_identifier** (*str*) – Trigger Identifier specifies the instance of the trigger you want to override. If trigger is specified as `NIDIGITAL_VAL_START_TRIGGER`, this parameter must be an empty string. If trigger is specified as `NIDIGITAL_VAL_CONDITIONAL_JUMP_TRIGGER`, allowed values are conditionalJumpTrigger0, conditionalJumpTrigger1, conditionalJumpTrigger2, and conditionalJumpTrigger3.

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.

tdr

`nidigital.Session.`**`tdr`**`(`*apply_offsets=True*`)`

Measures propagation delays through cables, connectors, and load boards using Time-Domain Reflectometry (TDR). Ensure that the channels and pins you select are connected to an open circuit.

Tip: This method can be called on specific channels within your `nidigital.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[ ... ].tdr()`

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

Example: `my_session.tdr()`

Parameters **apply_offsets** (*bool*) – A Boolean that specifies whether to apply the measured TDR offsets. If you need to adjust the measured offsets prior to applying, set this input to False, and call the `nidigital.Session.apply_tdr_offsets()` method to specify the adjusted TDR offsets values.

Return type list of hightime.timedelta

Returns Measured TDR offsets specified in seconds.

unload_all_patterns

`nidigital.Session.`**`unload_all_patterns`**`(`*unload_keep_alive_pattern=False*`)`

Unloads all patterns, source waveforms, and capture waveforms from a digital pattern instrument.

Parameters **unload_keep_alive_pattern** (*bool*) – A Boolean that specifies whether to keep or unload the keep alive pattern.

unload_specifications

`nidigital.Session.`**`unload_specifications`**`(`*file_paths*`)`

Unloads the given specifications sheets present in the previously loaded specifications files that you select.

You must call `nidigital.Session.load_specifications_levels_and_timing()` to reload the files with updated specifications values. You must then call `nidigital.Session.apply_levels_and_timing()` in order to apply the levels and timing values that reference the updated specifications values.

Parameters **`file_paths`** (*`str or basic sequence of str`*) – Absolute file path of one or more loaded specifications files.

unlock

`nidigital.Session.`**`unlock`**`()`

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

wait_until_done

`nidigital.Session.`**`wait_until_done`**`(`*timeout=hightime.timedelta(seconds=10.0)*`)`

Waits until the pattern burst has completed or the timeout has expired.

Parameters **`timeout`** (*`hightime.timedelta, datetime.timedelta, or float in seconds`*) – Maximum time (in seconds) allowed for this method to complete. If this method does not complete within this time interval, this method returns an error.

write_sequencer_flag

`nidigital.Session.`**`write_sequencer_flag`**`(`*flag*, *value*`)`

Writes the state of a pattern sequencer flag. Use pattern sequencer flags to coordinate execution between the pattern sequencer and a runtime test program.

Parameters

- **`flag`** (*`nidigital.SequencerFlag`*) – The pattern sequencer flag to write.
  - *`FLAG0`* (“seqflag0”): Writes pattern sequencer flag 0.
  - *`FLAG1`* (“seqflag1”): Writes pattern sequencer flag 1.
  - *`FLAG2`* (“seqflag2”): Writes pattern sequencer flag 2.
  - *`FLAG3`* (“seqflag3”): Writes pattern sequencer flag 3.
- **`value`** (*`bool`*) – A Boolean that assigns a state to the pattern sequencer flag you specify.

write_sequencer_register

`nidigital.Session.`**`write_sequencer_register`**`(`*reg*, *value*`)`

Writes a value to a pattern sequencer register. Use pattern sequencer registers to pass numeric values between the pattern sequencer and a runtime test program.

Parameters

- **`reg`** (*`nidigital.SequencerRegister`*) – The sequencer register you want to write to.

  - *REGISTER0* ("reg0"): Writes sequencer register 0.
  - *REGISTER1* ("reg1"): Writes sequencer register 1.
  - *REGISTER2* ("reg2"): Writes sequencer register 2.
  - *REGISTER3* ("reg3"): Writes sequencer register 3.
  - *REGISTER4* ("reg4"): Writes sequencer register 4.
  - *REGISTER5* ("reg5"): Writes sequencer register 5.
  - *REGISTER6* ("reg6"): Writes sequencer register 6.
  - *REGISTER7* ("reg7"): Writes sequencer register 7.
  - *REGISTER8* ("reg8"): Writes sequencer register 8.
  - *REGISTER9* ("reg9"): Writes sequencer register 9.
  - *REGISTER10* ("reg10"): Writes sequencer register 10.
  - *REGISTER11* ("reg11"): Writes sequencer register 11.
  - *REGISTER12* ("reg12"): Writes sequencer register 12.
  - *REGISTER13* ("reg13"): Writes sequencer register 13.
  - *REGISTER14* ("reg14"): Writes sequencer register 14.
  - *REGISTER15* ("reg15"): Writes sequencer register 15.
- **value** (*int*) – The value you want to write to the register.

write_source_waveform_broadcast

nidigital.Session.**write_source_waveform_broadcast**(*waveform_name*, *waveform_data*)

Writes the same waveform data to all sites. Use this write method if you set the data_mapping parameter of the create source waveform method to *BROADCAST*.

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **waveform_data** (*list of int*) – 1D array of samples to use as source data to apply to all sites.

write_source_waveform_data_from_file_tdms

nidigital.Session.**write_source_waveform_data_from_file_tdms**(*waveform_name*, *waveform_file_path*)

Writes a source waveform based on the waveform data and configuration information the file contains.

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.

- **waveform_file_path** (*str*) – Absolute file path to the load source waveform file (.tdms).

write_source_waveform_site_unique

`nidigital.Session.`**`write_source_waveform_site_unique`**(*waveform_name*, *waveform_data*)

Writes one waveform per site. Use this write method if you set the parameter of the create source waveform method to Site Unique.

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **waveform_data** (*{ int: basic sequence of unsigned int, int: basic sequence of unsigned int, .. }*) – Dictionary where each key is a site number and value is a collection of samples to use as source data

write_static

`nidigital.Session.`**`write_static`**(*state*)

Writes a static state to the specified pins. The selected pins remain in the specified state until the next pattern burst or call to this method. If there are uncommitted changes to levels or the termination mode, this method commits the changes to the pins. This method does not change the selected pin method. If you write a static state to a pin that does not have the Digital method selected, the new static state is stored by the instrument, and affects the state of the pin the next time you change the selected method to Digital.

Tip: This method can be called on specific channels within your `nidigital.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[ ... ].write_static()`

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

Example: `my_session.write_static()`

Parameters **state** (*nidigital.WriteStaticPinState*) – Parameter that specifies one of the following digital states to assign to the pin.

- *ZERO*: Specifies to drive low.
- *ONE*: Specifies to drive high.
- *X*: Specifies to not drive.

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.

Properties

active_load_ioh

nidigital.Session.**active_load_ioh**

Specifies the current that the DUT sources to the active load while outputting a voltage above VCOM.

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

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

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

Example: `my_session.active_load_ioh`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_ACTIVE_LOAD_IOH**

---

active_load_iol

nidigital.Session.**active_load_iol**

Specifies the current that the DUT sinks from the active load while outputting a voltage below VCOM.

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

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

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

Example: `my_session.active_load_iol`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_ACTIVE_LOAD_IOL**

active_load_vcom

`nidigital.Session.`**`active_load_vcom`**

Specifies the voltage level at which the active load circuit switches between sourcing current and sinking current.

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

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

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

Example: `my_session.active_load_vcom`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_ACTIVE_LOAD_VCOM**

cache

`nidigital.Session.`**`cache`**

Specifies whether to cache the value of properties. When caching is enabled, the instrument driver keeps track of the current instrument settings and avoids sending redundant commands to the instrument. This significantly increases execution speed. Caching is always enabled in the driver, regardless of the value of this property.

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: **NIDIGITAL_ATTR_CACHE**

channel_count

`nidigital.Session.`**`channel_count`**

Returns the number of channels that the specific digital pattern instrument driver supports.

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:

- C Attribute: **NIDIGITAL_ATTR_CHANNEL_COUNT**

clock_generator_frequency

`nidigital.Session.`**`clock_generator_frequency`**

Specifies the frequency for the clock generator.

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

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

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

Example: `my_session.clock_generator_frequency`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_CLOCK_GENERATOR_FREQUENCY**

clock_generator_is_running

`nidigital.Session.`**`clock_generator_is_running`**

Indicates whether the clock generator is running.

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

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

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

Example: `my_session.clock_generator_is_running`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read only
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_CLOCK_GENERATOR_IS_RUNNING**

conditional_jump_trigger_terminal_name

`nidigital.Session.`**`conditional_jump_trigger_terminal_name`**

Specifies the terminal name from which the exported conditional jump trigger signal may be routed to other instruments through the PXI trigger bus. You can use this signal to trigger other instruments when the conditional jump trigger instance asserts on the digital pattern instrument.

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

Example: `my_session.conditional_jump_triggers[ ... ].conditional_jump_trigger_terminal_name`

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

Example: `my_session.conditional_jump_trigger_terminal_name`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_CONDITIONAL_JUMP_TRIGGER_TERMINAL_NAME**

conditional_jump_trigger_type

`nidigital.Session.`**`conditional_jump_trigger_type`**

Disables the conditional jump trigger or configures it for either hardware triggering or software triggering. The default value is *`NONE`*.

Valid Values:	
`NONE`	Disables the conditional jump trigger.
`DIGITAL_EDGE`	Configures the conditional jump trigger for hardware triggering.
`SOFTWARE`	Configures the conditional jump trigger for software triggering.

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

Example: `my_session.conditional_jump_triggers[ ... ].conditional_jump_trigger_type`

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

Example: `my_session.conditional_jump_trigger_type`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TriggerType
Permissions	read-write
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_CONDITIONAL_JUMP_TRIGGER_TYPE**

cycle_number_history_ram_trigger_cycle_number

`nidigital.Session.`**`cycle_number_history_ram_trigger_cycle_number`**

Specifies the cycle number on which History RAM starts acquiring pattern information when configured for a cycle number trigger.

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: **NIDIGITAL_ATTR_CYCLE_NUMBER_HISTORY_RAM_TRIGGER_CYCLE_NUMBER**

digital_edge_conditional_jump_trigger_edge

`nidigital.Session.`**`digital_edge_conditional_jump_trigger_edge`**

Configures the active edge of the incoming trigger signal for the conditional jump trigger instance. The default value is *`RISING`*.

Valid Values:	
`RISING`	Specifies the signal transition from low level to high level.
`FALLING`	Specifies the signal transition from high level to low level.

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

Example: `my_session.conditional_jump_triggers[ ... ].digital_edge_conditional_jump_trigger_edge`

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

Example: `my_session.digital_edge_conditional_jump_trigger_edge`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.DigitalEdge
Permissions	read-write
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_CONDITIONAL_JUMP_TRIGGER_EDGE**

digital_edge_conditional_jump_trigger_source

`nidigital.Session.`**`digital_edge_conditional_jump_trigger_source`**

Configures the digital trigger source terminal for a conditional jump trigger instance. The PXIe-6570/6571 supports triggering through the PXI trigger bus. You can specify source terminals in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The source terminal can also be a terminal from another device, in which case the NI-Digital Pattern Driver automatically finds a route (if one is available) from that terminal to the input terminal (going through a physical PXI backplane trigger line). For example, you can set the source terminal on Dev1 to be /Dev2/ConditionalJumpTrigger0. The default value is VI_NULL.

Valid Values:
String identifier to any valid terminal name

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

Example: `my_session.conditional_jump_triggers[ ... ].digital_edge_conditional_jump_trigger_source`

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

Example: `my_session.digital_edge_conditional_jump_trigger_source`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_CONDITIONAL_JUMP_TRIGGER_SOURCE**

digital_edge_start_trigger_edge

`nidigital.Session.`**`digital_edge_start_trigger_edge`**

Specifies the active edge for the Start trigger. This property is used when the `nidigital.Session.start_trigger_type` property is set to Digital Edge.

Defined Values:	
RISING	Asserts the trigger when the signal transitions from low level to high level.
FALLING	Asserts the trigger when the signal transitions from high level to low level.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_START_TRIGGER_EDGE**

digital_edge_start_trigger_source

`nidigital.Session.`**`digital_edge_start_trigger_source`**

Specifies the source terminal for the Start trigger. This property is used when the *`nidigital.Session.start_trigger_type`* property is set to Digital Edge. You can specify source terminals in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The source terminal can also be a terminal from another device, in which case the NI-Digital Pattern Driver automatically finds a route (if one is available) from that terminal to the input terminal (going through a physical PXI backplane trigger line). For example, you can set the source terminal on Dev1 to be /Dev2/StartTrigger.

Defined Values:	
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

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:

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_START_TRIGGER_SOURCE**

driver_setup

`nidigital.Session.`**`driver_setup`**

This property returns initial values for NI-Digital Pattern Driver properties as a string.

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: **NIDIGITAL_ATTR_DRIVER_SETUP**

exported_conditional_jump_trigger_output_terminal

`nidigital.Session.`**`exported_conditional_jump_trigger_output_terminal`**

Specifies the terminal to output the exported signal of the specified instance of the conditional jump trigger. The default value is VI_NULL.

Valid Values:	
VI_NULL ("")	Returns an empty string
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

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

Example: `my_session.conditional_jump_triggers[ ... ].exported_conditional_jump_trigger_output_terminal`

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

Example: `my_session.exported_conditional_jump_trigger_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_EXPORTED_CONDITIONAL_JUMP_TRIGGER_OUTPUT_TERMINAL**

exported_pattern_opcode_event_output_terminal

`nidigital.Session.`**`exported_pattern_opcode_event_output_terminal`**

Specifies the destination terminal for exporting the Pattern Opcode Event. Terminals can be specified in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Defined Values:	
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

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

Example: `my_session.pattern_opcode_events[ ... ].exported_pattern_opcode_event_output_terminal`

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

Example: `my_session.exported_pattern_opcode_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	pattern_opcode_events

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

- C Attribute: **NIDIGITAL_ATTR_EXPORTED_PATTERN_OPCODE_EVENT_OUTPUT_TERMINAL**

exported_start_trigger_output_terminal

`nidigital.Session.`**`exported_start_trigger_output_terminal`**

Specifies the destination terminal for exporting the Start trigger. Terminals can be specified in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Defined Values:	
Do not export signal	The signal is not exported.
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

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:

- C Attribute: **NIDIGITAL_ATTR_EXPORTED_START_TRIGGER_OUTPUT_TERMINAL**

frequency_counter_hysteresis_enabled

`nidigital.Session.`**`frequency_counter_hysteresis_enabled`**

Specifies whether hysteresis is enabled for the frequency counters of the digital pattern instrument.

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: **NIDIGITAL_ATTR_FREQUENCY_COUNTER_HYSTERESIS_ENABLED**

frequency_counter_measurement_mode

`nidigital.Session.`**`frequency_counter_measurement_mode`**

Determines how the frequency counters of the digital pattern instrument make measurements.

Valid Values:	
BANKED	Each discrete frequency counter is mapped to specific channels and makes frequency measurements from only those channels. Use banked mode when you need access to the full measure frequency range of the instrument. **Note:** If you request frequency measurements from multiple channels within the same bank, the measurements are made in series for the channels in that bank.
PARALLEL	All discrete frequency counters make frequency measurements from all channels in parallel with one another. Use parallel mode to increase the speed of frequency measurements if you do not need access to the full measure frequency range of the instrument; in parallel mode, you can also add *nidigital.Session.frequency_counter_hysteresis_enabled* to reduce measurement noise.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_FREQUENCY_COUNTER_MEASUREMENT_MODE**

frequency_counter_measurement_time

`nidigital.Session.`**`frequency_counter_measurement_time`**

Specifies the measurement time for the frequency counter.

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

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

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

Example: `my_session.frequency_counter_measurement_time`

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_FREQUENCY_COUNTER_MEASUREMENT_TIME**

group_capabilities

`nidigital.Session.`**`group_capabilities`**

Returns a string that contains a comma-separated list of class-extension groups that the driver implements.

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: **NIDIGITAL_ATTR_GROUP_CAPABILITIES**

halt_on_keep_alive_opcode

`nidigital.Session.`**`halt_on_keep_alive_opcode`**

Specifies whether keep_alive opcodes should behave like halt opcodes.

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: **NIDIGITAL_ATTR_HALT_ON_KEEP_ALIVE_OPCODE**

history_ram_buffer_size_per_site

`nidigital.Session.`**`history_ram_buffer_size_per_site`**

Specifies the size, in samples, of the host memory buffer. The default value is 32000.

Valid Values:
0-INT64_MAX

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: **NIDIGITAL_ATTR_HISTORY_RAM_BUFFER_SIZE_PER_SITE**

history_ram_cycles_to_acquire

`nidigital.Session.`**`history_ram_cycles_to_acquire`**

Configures which cycles History RAM acquires after the trigger conditions are met. If you configure History RAM to only acquire failed cycles, you must set the pretrigger samples for History RAM to 0.

Defined Values:	
FAILED	Only acquires cycles that fail a compare after the triggering conditions are met.
ALL	Acquires all cycles after the triggering conditions are met.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_HISTORY_RAM_CYCLES_TO_ACQUIRE**

history_ram_max_samples_to_acquire_per_site

`nidigital.Session.`**`history_ram_max_samples_to_acquire_per_site`**

Specifies the maximum number of History RAM samples to acquire per site. If the property is set to -1, it will acquire until the History RAM buffer is full.

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: **NIDIGITAL_ATTR_HISTORY_RAM_MAX_SAMPLES_TO_ACQUIRE_PER_SITE**

history_ram_number_of_samples_is_finite

`nidigital.Session.`**`history_ram_number_of_samples_is_finite`**

Specifies whether the instrument acquires a finite number of History Ram samples or acquires continuously. The maximum number of samples that will be acquired when this property is set to True is determined by the instrument History RAM depth specification and the History RAM Max Samples to Acquire Per Site property. The default value is True.

Valid Values:	
True	Specifies that History RAM results will not stream into the host buffer until a History RAM fetch API is called.
False	Specifies that History RAM results will automatically start streaming into a host buffer after a pattern is burst and the History RAM has triggered.

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: **NIDIGITAL_ATTR_HISTORY_RAM_NUMBER_OF_SAMPLES_IS_FINITE**

history_ram_pretrigger_samples

`nidigital.Session.`**`history_ram_pretrigger_samples`**

Specifies the number of samples to acquire before the trigger conditions are met. If you configure History RAM to only acquire failed cycles, you must set the pretrigger samples for History RAM to 0.

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: **NIDIGITAL_ATTR_HISTORY_RAM_PRETRIGGER_SAMPLES**

history_ram_trigger_type

`nidigital.Session.`**`history_ram_trigger_type`**

Specifies the type of trigger condition on which History RAM starts acquiring pattern information.

Defined Values:	
FIRST_FAILURE	Starts acquiring pattern information in History RAM on the first failed cycle in a pattern burst.
CYCLE_NUMBER	Starts acquiring pattern information in History RAM starting from a specified cycle number.
PATTERN_LABEL	Starts acquiring pattern information in History RAM starting from a specified pattern label, augmented by vector and cycle offsets.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_HISTORY_RAM_TRIGGER_TYPE**

instrument_firmware_revision

`nidigital.Session.`**`instrument_firmware_revision`**

Returns a string that contains the firmware revision information for the digital pattern instrument.

Tip: This property can be set/get on specific instruments within your *`nidigital.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 *`nidigital.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:

- C Attribute: **NIDIGITAL_ATTR_INSTRUMENT_FIRMWARE_REVISION**

instrument_manufacturer

`nidigital.Session.`**`instrument_manufacturer`**

Returns a string ("National Instruments") that contains the name of the manufacturer of the digital pattern 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:

- C Attribute: **NIDIGITAL_ATTR_INSTRUMENT_MANUFACTURER**

instrument_model

`nidigital.Session.`**`instrument_model`**

Returns a string that contains the model number or name of the digital pattern 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:

- C Attribute: **NIDIGITAL_ATTR_INSTRUMENT_MODEL**

interchange_check

`nidigital.Session.`**`interchange_check`**

This property is not supported.

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: **NIDIGITAL_ATTR_INTERCHANGE_CHECK**

io_resource_descriptor

`nidigital.Session.`**`io_resource_descriptor`**

Returns a string that contains the resource descriptor that the NI-Digital Pattern Driver uses to identify the digital pattern 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:

- C Attribute: **NIDIGITAL_ATTR_IO_RESOURCE_DESCRIPTOR**

is_keep_alive_active

`nidigital.Session.`**`is_keep_alive_active`**

Returns True if the digital pattern instrument is driving the keep alive pattern.

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:

- C Attribute: **NIDIGITAL_ATTR_IS_KEEP_ALIVE_ACTIVE**

logical_name

`nidigital.Session.`**`logical_name`**

Returns a string containing the logical name that you specified when opening the current IVI session. This property is not supported.

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: **NIDIGITAL_ATTR_LOGICAL_NAME**

mask_compare

`nidigital.Session.`**`mask_compare`**

Specifies whether the pattern comparisons are masked or not. When set to True for a specified pin, failures on that pin will be masked.

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

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

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

Example: `my_session.mask_compare`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_MASK_COMPARE**

pattern_label_history_ram_trigger_cycle_offset

`nidigital.Session.`**`pattern_label_history_ram_trigger_cycle_offset`**

Specifies the number of cycles that follow the specified pattern label and vector offset, after which History RAM will start acquiring pattern information when configured for a pattern label trigger.

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: **NIDIGITAL_ATTR_PATTERN_LABEL_HISTORY_RAM_TRIGGER_CYCLE_OFFSET**

pattern_label_history_ram_trigger_label

`nidigital.Session.`**`pattern_label_history_ram_trigger_label`**

Specifies the pattern label, augmented by the vector and cycle offset, to determine the point where History RAM will start acquiring pattern information when configured for a pattern label trigger.

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:

- C Attribute: **NIDIGITAL_ATTR_PATTERN_LABEL_HISTORY_RAM_TRIGGER_LABEL**

pattern_label_history_ram_trigger_vector_offset

`nidigital.Session.`**`pattern_label_history_ram_trigger_vector_offset`**

Specifies the number of vectors that follow the specified pattern label, after which History RAM will start acquiring pattern information when configured for a pattern label trigger.

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: **NIDIGITAL_ATTR_PATTERN_LABEL_HISTORY_RAM_TRIGGER_VECTOR_OFFSET**

pattern_opcode_event_terminal_name

`nidigital.Session.`**`pattern_opcode_event_terminal_name`**

Specifies the terminal name for the output trigger signal of the specified instance of a Pattern Opcode Event. You can use this terminal name as an input signal source for another trigger.

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

Example: `my_session.pattern_opcode_events[ ... ].pattern_opcode_event_terminal_name`

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

Example: `my_session.pattern_opcode_event_terminal_name`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	pattern_opcode_events

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

- C Attribute: **NIDIGITAL_ATTR_PATTERN_OPCODE_EVENT_TERMINAL_NAME**

ppmu_allow_extended_voltage_range

`nidigital.Session.`**`ppmu_allow_extended_voltage_range`**

Enables the instrument to operate in additional voltage ranges where instrument specifications may differ from standard ranges. When set to True, this property enables extended voltage range operation. Review specification deviations for application suitability before using this property. NI recommends setting this property to False when not using the extended voltage range to avoid unintentional use of this range. The extended voltage range is supported only for PPMU, with the output method set to DC Voltage. A voltage glitch may occur when you change the PPMU output voltage from a standard range to the extended voltage range, or vice-versa, while the PPMU is sourcing. NI recommends temporarily changing the `nidigital.Session.selected_function` property to Off before sourcing a voltage level that requires a range change.

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

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

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

Example: `my_session.ppmu_allow_extended_voltage_range`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_ALLOW_EXTENDED_VOLTAGE_RANGE**

ppmu_aperture_time

`nidigital.Session.`**`ppmu_aperture_time`**

Specifies the measurement aperture time for the PPMU. The *`nidigital.Session.ppmu_aperture_time_units`* property sets the units of the PPMU aperture time.

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

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

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

Example: `my_session.ppmu_aperture_time`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_APERTURE_TIME**

ppmu_aperture_time_units

`nidigital.Session.`**`ppmu_aperture_time_units`**

Specifies the units of the measurement aperture time for the PPMU.

Defined Values:	
SECONDS	Specifies the aperture time in seconds.

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

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

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

Example: `my_session.ppmu_aperture_time_units`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.PPMUApertureTimeUnits
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_APERTURE_TIME_UNITS**

ppmu_current_level

nidigital.Session.**ppmu_current_level**

Specifies the current level, in amps, that the PPMU forces to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Current. Specify valid values for the current level using the `nidigital.Session.PPMU_ConfigureCurrentLevelRange()` method.

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 or pins within your *nidigital.Session* instance. Use Python index notation on the repeated capabilities container channels or pins to specify a subset.

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

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

Example: `my_session.ppmu_current_level`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LEVEL**

ppmu_current_level_range

`nidigital.Session.`**`ppmu_current_level_range`**

Specifies the range of valid values for the current level, in amps, that the PPMU forces to the DUT. This property is applicable only when you set the *`nidigital.Session.ppmu_output_function`* property to DC Current.

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

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

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

Example: `my_session.ppmu_current_level_range`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LEVEL_RANGE**

---

ppmu_current_limit

`nidigital.Session.`**`ppmu_current_limit`**

Specifies the current limit, in amps, that the output cannot exceed while the PPMU forces voltage to the DUT. This property is applicable only when you set the *`nidigital.Session.ppmu_output_function`* property to DC Voltage. The PXIe-6570/6571 does not support the *`nidigital.Session.ppmu_current_limit`* property and only allows configuration of the *`nidigital.Session.ppmu_current_limit_range`* property.

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

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

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

Example: `my_session.ppmu_current_limit`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LIMIT**

ppmu_current_limit_behavior

nidigital.Session.**ppmu_current_limit_behavior**

Specifies how the output should behave when the current limit is reached.

Defined Values:	
REGULATE	Controls output current so that it does not exceed the current limit. Power continues to generate even if the current limit is reached.

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

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

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

Example: `my_session.ppmu_current_limit_behavior`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.PPMUCurrentLimitBehavior
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LIMIT_BEHAVIOR**

ppmu_current_limit_range

nidigital.Session.**ppmu_current_limit_range**

Specifies the valid range, in amps, to which the current limit can be set while the PPMU forces voltage to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Voltage.

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

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

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

Example: `my_session.ppmu_current_limit_range`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LIMIT_RANGE**

ppmu_current_limit_supported

`nidigital.Session.`**`ppmu_current_limit_supported`**

Returns whether the device supports configuration of a current limit when you set the *nidigital.Session.ppmu_output_function* property to DC Voltage.

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

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

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

Example: `my_session.ppmu_current_limit_supported`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read only
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LIMIT_SUPPORTED**

ppmu_output_function

nidigital.Session.**ppmu_output_function**

Specifies whether the PPMU forces voltage or current to the DUT.

Defined Values:	
VOLTAGE	Specifies the output method to DC Voltage.
CURRENT	Specifies the output method to DC Current.

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

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

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

Example: `my_session.ppmu_output_function`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.PPMUOutputFunction
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_OUTPUT_FUNCTION**

ppmu_voltage_level

nidigital.Session.**ppmu_voltage_level**

Specifies the voltage level, in volts, that the PPMU forces to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Voltage.

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

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

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

Example: `my_session.ppmu_voltage_level`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_VOLTAGE_LEVEL**

ppmu_voltage_limit_high

`nidigital.Session.`**`ppmu_voltage_limit_high`**

Specifies the maximum voltage limit, or high clamp voltage (V $_{CH}$), in volts, at the pin when the PPMU forces current to the DUT. This property is applicable only when you set the *`nidigital.Session.ppmu_output_function`* property to DC Current.

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

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

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

Example: `my_session.ppmu_voltage_limit_high`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_VOLTAGE_LIMIT_HIGH**

ppmu_voltage_limit_low

`nidigital.Session.`**`ppmu_voltage_limit_low`**

Specifies the minimum voltage limit, or low clamp voltage (V $_{CL}$), in volts, at the pin when the PPMU forces current to the DUT. This property is applicable only when you set the *`nidigital.Session.ppmu_output_function`* property to DC Current.

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

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

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

Example: `my_session.ppmu_voltage_limit_low`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_VOLTAGE_LIMIT_LOW**

query_instrument_status

nidigital.Session.**query_instrument_status**

Specifies whether the NI-Digital Pattern Driver queries the digital pattern instrument status after each operation. The instrument status is always queried, regardless of the property setting.

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: **NIDIGITAL_ATTR_QUERY_INSTRUMENT_STATUS**

range_check

nidigital.Session.**range_check**

Checks the range and validates parameter and property values you pass to NI-Digital Pattern Driver methods. Ranges are always checked, regardless of the property setting.

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: **NIDIGITAL_ATTR_RANGE_CHECK**

record_coercions

`nidigital.Session.`**`record_coercions`**

Specifies whether the IVI engine keeps a list of the value coercions it makes for integer and real type properties. Enabling record value coercions is not supported.

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: **NIDIGITAL_ATTR_RECORD_COERCIONS**

selected_function

`nidigital.Session.`**`selected_function`**

Caution: In the Disconnect state, some I/O protection and sensing circuitry remains exposed. Do not subject the instrument to voltage beyond its operating range.

Specifies whether digital pattern instrument channels are controlled by the pattern sequencer or PPMU, disconnected, or off.

Defined Values:	
DIGITAL	The pin is connected to the driver, comparator, and active load methods. The PPMU is not sourcing, but can make voltage measurements. The state of the digital pin driver when you change the *nidigital.Session.selected_function* to Digital is determined by the most recent call to the *nidigital.Session.write_static()* method or the last vector of the most recently executed pattern burst, whichever happened last. Use the *nidigital.Session.write_static()* method to control the state of the digital pin driver through software. Use the *nidigital.Session.burst_pattern()* method to control the state of the digital pin driver through a pattern. Set the **selectDigitalFunction** parameter of the *nidigital.Session.burst_pattern()* method to True to automatically switch the *nidigital.Session.selected_function* of the pins in the pattern burst to *DIGITAL*.
PPMU	The pin is connected to the PPMU. The driver, comparator, and active load are off while this method is selected. Call the *nidigital.Session.ppmu_source()* method to source a voltage or current. The *nidigital.Session.ppmu_source()* method automatically switches the *nidigital.Session.selected_function* to the PPMU state and starts sourcing from the PPMU. Changing the *nidigital.Session.selected_function* to *DISCONNECT*, *OFF*, or *DIGITAL* causes the PPMU to stop sourcing. If you set the *nidigital.Session.selected_function* property to PPMU, the PPMU is initially not sourcing.
OFF	The pin is electrically connected, and the PPMU and digital pin driver are off while this method is selected.
DISCONNECT	The pin is electrically disconnected from instrument methods. Selecting this method causes the PPMU to stop sourcing prior to disconnecting the pin.

Note: You can make PPMU voltage measurements using the *nidigital.Session.ppmu_measure()* method from within any *nidigital.Session.selected_function*.

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

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

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

Example: `my_session.selected_function`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.SelectedFunction
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_SELECTED_FUNCTION**

sequencer_flag_terminal_name

`nidigital.Session.`**`sequencer_flag_terminal_name`**

Specifies the terminal name for the output trigger signal of the Sequencer Flags trigger. You can use this terminal name as an input signal 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:

- C Attribute: **NIDIGITAL_ATTR_SEQUENCER_FLAG_TERMINAL_NAME**

serial_number

`nidigital.Session.`**`serial_number`**

Returns the serial number of the device.

Tip: This property can be set/get on specific instruments within your *`nidigital.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 *`nidigital.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:

- C Attribute: **NIDIGITAL_ATTR_SERIAL_NUMBER**

simulate

`nidigital.Session.`**`simulate`**

Simulates I/O operations. After you open a session, you cannot change the simulation state. Use the `nidigital.Session.__init__()` method to enable simulation.

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: **NIDIGITAL_ATTR_SIMULATE**

specific_driver_class_spec_major_version

`nidigital.Session.`**`specific_driver_class_spec_major_version`**

Returns the major version number of the class specification with which NI-Digital is compliant. This property is not supported.

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:

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MAJOR_VERSION**

specific_driver_class_spec_minor_version

`nidigital.Session.`**`specific_driver_class_spec_minor_version`**

Returns the minor version number of the class specification with which NI-Digital is compliant. This property is not supported.

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:

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MINOR_VERSION**

specific_driver_description

`nidigital.Session.`**`specific_driver_description`**

Returns a string that contains a brief description of the NI-Digital Pattern 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:

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_DESCRIPTION**

specific_driver_prefix

`nidigital.Session.`**`specific_driver_prefix`**

Returns a string that contains the prefix for the NI-Digital Pattern 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:

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_PREFIX**

specific_driver_revision

`nidigital.Session.`**`specific_driver_revision`**

Returns a string that contains additional version information about the NI-Digital Pattern Driver. For example, the driver can return Driver: NI-Digital 16.0 as the value of this property.

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: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_REVISION**

specific_driver_vendor

nidigital.Session.**specific_driver_vendor**

Returns a string ("National Instruments") that contains the name of the vendor that supplies the NI-Digital Pattern 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:

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_VENDOR**

start_label

nidigital.Session.**start_label**

Specifies the pattern name or exported pattern label from which to start bursting the pattern.

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:

- C Attribute: **NIDIGITAL_ATTR_START_LABEL**

start_trigger_terminal_name

nidigital.Session.**start_trigger_terminal_name**

Specifies the terminal name for the output trigger signal of the Start trigger. You can use this terminal name as an input signal 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:

- C Attribute: **NIDIGITAL_ATTR_START_TRIGGER_TERMINAL_NAME**

start_trigger_type

nidigital.Session.**start_trigger_type**

Specifies the Start trigger type. The digital pattern instrument waits for this trigger after you call the `nidigital.Session.init()` method or the *nidigital.Session.burst_pattern()* method, and does not burst a pattern until this trigger is received.

Defined Values:	
NONE	Disables the Start trigger. Pattern bursting starts immediately after you call the `nidigital.Session.init()` method or the *nidigital.Session.burst_pattern()* method.
DIGITAL_EDGE	Pattern bursting does not start until the digital pattern instrument detects a digital edge.
SOFTWARE	Pattern bursting does not start until the digital pattern instrument receives a software Start trigger. Create a software Start trigger by calling the *nidigital.Session.send_software_edge_trigger()* method and selecting start trigger in the **trigger** parameter.Related information: SendSoftwareEdgeTrigger method.

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.TriggerType
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_START_TRIGGER_TYPE**

supported_instrument_models

nidigital.Session.**supported_instrument_models**

Returns a comma delimited string that contains the supported digital pattern instrument models for 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:

- C Attribute: **NIDIGITAL_ATTR_SUPPORTED_INSTRUMENT_MODELS**

tdr_endpoint_termination

`nidigital.Session.`**`tdr_endpoint_termination`**

Specifies whether TDR Channels are connected to an open circuit or a short to ground.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_TDR_ENDPOINT_TERMINATION**

tdr_offset

`nidigital.Session.`**`tdr_offset`**

Specifies the TDR Offset.

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

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

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

Example: `my_session.tdr_offset`

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, pins

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

- C Attribute: **NIDIGITAL_ATTR_TDR_OFFSET**

termination_mode

`nidigital.Session.`**`termination_mode`**

Specifies the behavior of the pin during non-drive cycles.

Defined Values:	
ACTIVE_LOAD	Specifies that, for non-drive pin states (L, H, X, V, M, E), the active load is connected and the instrument sources or sinks a defined amount of current to load the DUT. The amount of current sourced by the instrument and therefore sunk by the DUT is specified by IOL. The amount of current sunk by the instrument and therefore sourced by the DUT is specified by IOH. The voltage at which the instrument changes between sourcing and sinking is specified by VCOM.
VTERM	Specifies that, for non-drive pin states (L, H, X, V, M, E), the pin driver terminates the pin to the configured VTERM voltage through a 50 Ω impedance. VTERM is adjustable to allow for the pin to terminate at a set level. This is useful for instruments that might operate incorrectly if an instrument pin is unterminated and is allowed to float to any voltage level within the instrument voltage range. To address this issue, enable VTERM by configuring the VTERM pin level to the desired voltage and selecting the VTERM termination mode. Setting VTERM to 0 V and selecting the VTERM termination mode has the effect of connecting a 50 Ω termination to ground, which provides an effective 50 Ω impedance for the pin. This can be useful for improving signal integrity of certain DUTs by reducing reflections while the DUT drives the pin.
HIGH_Z	Specifies that, for non-drive pin states (L, H, X, V, M, E), the pin driver is put in a high-impedance state and the active load is disabled.

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

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

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

Example: `my_session.termination_mode`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TerminationMode
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_TERMINATION_MODE**

timing_absolute_delay

nidigital.Session.**timing_absolute_delay**

Specifies a timing delay, measured in seconds, and applies the delay to the digital pattern instrument in addition to TDR and calibration adjustments. If the *nidigital.Session.timing_absolute_delay_enabled* property is set to True, this value is the intermodule skew measured by NI-TClk. You can modify this value to override the timing delay and align the I/O timing of this instrument with another instrument that shares the same reference clock. If the *nidigital.Session.timing_absolute_delay_enabled* property is False, this property will return 0.0. Changing the *nidigital.Session.timing_absolute_delay_enabled* property from False to True will set the *nidigital.Session.timing_absolute_delay* value back to your previously set value.

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

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

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

Example: `my_session.timing_absolute_delay`

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	instruments

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

- C Attribute: **NIDIGITAL_ATTR_TIMING_ABSOLUTE_DELAY**

timing_absolute_delay_enabled

nidigital.Session.**timing_absolute_delay_enabled**

Specifies whether the *nidigital.Session.timing_absolute_delay* property should be applied to adjust the digital pattern instrument timing reference relative to other instruments in the system. Do not use this feature with digital pattern instruments in a Semiconductor Test System (STS). Timing absolute delay conflicts with the adjustment performed during STS timing calibration. When set to True, the digital pattern instrument automatically adjusts the timing absolute delay to correct the instrument timing reference relative to other instruments in the system for better timing alignment among synchronized instruments.

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: **NIDIGITAL_ATTR_TIMING_ABSOLUTE_DELAY_ENABLED**

vih

`nidigital.Session.`**`vih`**

Specifies the voltage that the digital pattern instrument will apply to the input of the DUT when the test instrument drives a logic high (1).

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

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

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

Example: `my_session.vih`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VIH**

vil

`nidigital.Session.`**`vil`**

Specifies the voltage that the digital pattern instrument will apply to the input of the DUT when the test instrument drives a logic low (0).

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

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

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

Example: `my_session.vil`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VIL**

voh

`nidigital.Session.`**`voh`**

Specifies the output voltage from the DUT above which the comparator on the digital pattern test instrument interprets a logic high (H).

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

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

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

Example: `my_session.voh`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VOH**

vol

`nidigital.Session.vol`

Specifies the output voltage from the DUT below which the comparator on the digital pattern test instrument interprets a logic low (L).

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

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

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

Example: `my_session.vol`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VOL**

vterm

`nidigital.Session.vterm`

Specifies the termination voltage the digital pattern instrument applies during non-drive cycles when the termination mode is set to V_{term}. The instrument applies the termination voltage through a 50 Ω parallel termination resistance.

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

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

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

Example: `my_session.vterm`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VTERM**

NI-TClk Support

`nidigital.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

- *create_capture_waveform_serial*
- *create_source_waveform_from_file_tdms*
- *create_source_waveform_parallel*
- *create_source_waveform_serial*
- *create_time_set*
- *delete_all_time_sets*
- *disable_sites*
- *enable_sites*
- *fetch_capture_waveform*
- *fetch_history_ram_cycle_information*
- *frequency_counter_measure_frequency*
- *get_channel_names*
- *get_fail_count*
- *get_history_ram_sample_count*
- *get_pattern_pin_names*
- *get_pin_results_pin_information*
- *get_site_pass_fail*
- *get_time_set_drive_format*
- *get_time_set_edge*
- *get_time_set_edge_multiplier*
- *get_time_set_period*
- *initiate*
- *is_done*
- *is_site_enabled*
- *load_pattern*
- *load_pin_map*
- *load_specifications_levels_and_timing*
- *lock*
- *ppmu_measure*
- *ppmu_source*
- *read_sequencer_flag*
- *read_sequencer_register*
- *read_static*
- *reset*
- *reset_device*

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 `niDigital_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

nidigital.Session.channels[]

```
session.channels['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

pins

nidigital.Session.pins[]

```
session.pins['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

instruments

nidigital.Session.instruments[]

```
session.instruments['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

pattern_opcode_events

nidigital.Session.pattern_opcode_events[]

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.pattern_opcode_events['0-2'].channel_enabled = True
```

passes a string of `'patternOpcodeEvent0, patternOpcodeEvent1, patternOpcodeEvent2'` to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.pattern_opcode_events['patternOpcodeEvent0-patternOpcodeEvent2'].
↪channel_enabled = True
```

passes a string of `'patternOpcodeEvent0, patternOpcodeEvent1, patternOpcodeEvent2'` to the set attribute function.

conditional_jump_triggers

nidigital.Session.conditional_jump_triggers[]

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.conditional_jump_triggers['0-2'].channel_enabled = True
```

passes a string of `'conditionalJumpTrigger0, conditionalJumpTrigger1, conditionalJumpTrigger2'` to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.conditional_jump_triggers['conditionalJumpTrigger0-
↪conditionalJumpTrigger2'].channel_enabled = True
```

passes a string of `'conditionalJumpTrigger0, conditionalJumpTrigger1, conditionalJumpTrigger2'` to the set attribute function.

sites

nidigital.Session.sites[]

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.sites['0-2'].channel_enabled = True
```

passes a string of `'site0, site1, site2'` to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.sites['site0-site2'].channel_enabled = True
```

passes a string of `'site0, site1, site2'` to the set attribute function.

Enums

Enums used in NI-Digital Pattern Driver

BitOrder

class nidigital.**BitOrder**

MSB

The most significant bit is first. The first bit is in the 2^n place, where n is the number of bits.

LSB

The least significant bit is first. The first bit is in the 2^0 place.

DigitalEdge

class nidigital.**DigitalEdge**

RISING
Asserts the trigger when the signal transitions from low level to high level.

FALLING
Asserts the trigger when the signal transitions from high level to low level.

DriveFormat

class nidigital.**DriveFormat**

NR
Drive format remains at logic level after each bit.

RL
Drive format returns to a logic level low after each bit.

RH
Drive format returns to a logic level high after each bit.

SBC
Drive format returns to the complement logic level of the bit after each bit.

FrequencyMeasurementMode

class nidigital.**FrequencyMeasurementMode**

BANKED
Frequency measurements are made serially for groups of channels associated with a single frequency counter for each group.

Maximum frequency measured: 200 MHz.

PARALLEL
Frequency measurements are made by multiple frequency counters in parallel.

Maximum frequency measured: 100 MHz.

HistoryRAMCyclesToAcquire

class nidigital.**HistoryRAMCyclesToAcquire**

FAILED
Acquires failed cycles.

ALL
Acquires all cycles.

HistoryRAMTriggerType

class nidigital.**HistoryRAMTriggerType**

FIRST_FAILURE
First Failure History RAM trigger

CYCLE_NUMBER
Cycle Number History RAM trigger.

PATTERN_LABEL
Pattern Label History RAM trigger

PPMUApertureTimeUnits

class nidigital.**PPMUApertureTimeUnits**

SECONDS
Unit in seconds.

PPMUCurrentLimitBehavior

class nidigital.**PPMUCurrentLimitBehavior**

REGULATE
Controls output current so that it does not exceed the current limit. Power continues to generate even if the current limit is reached.

PPMUMeasurementType

class nidigital.**PPMUMeasurementType**

CURRENT
The PPMU measures current.

VOLTAGE
The PPMU measures voltage.

PPMUOutputFunction

class nidigital.**PPMUOutputFunction**

VOLTAGE
The PPMU forces voltage to the DUT.

CURRENT
The PPMU forces current to the DUT.

PinState

class nidigital.**PinState**

ZERO
A digital state of 0.

ONE
A digital state of 1.

L
A digital state of L (low).

H
A digital state of H (high).

X
A digital state of X (non-drive state).

M
A digital state of M (midband).

V
A digital state of V (compare high or low, not midband; store results from capture functionality if configured).

D
A digital state of D (drive data from source functionality if configured).

E
A digital state of E (compare data from source functionality if configured).

NOT_A_PIN_STATE
Not a pin state is used for non-existent DUT cycles.

PIN_STATE_NOT_ACQUIRED
Pin state could not be acquired because none of the pins mapped to the instrument in a multi-instrument session had any failures.

SelectedFunction

class nidigital.**SelectedFunction**

DIGITAL
The pattern sequencer controls the specified pin(s). If a pattern is currently bursting, the pin immediately switches to bursting the pattern. This option disconnects the PPMU.

PPMU
The PPMU controls the specified pin(s) and connects the PPMU. The pin driver is in a non-drive state, and the active load is disabled. The PPMU does not start sourcing or measuring until Source or Measure(PpmuMeasurementType) is called.

OFF
Puts the digital driver in a non-drive state, disables the active load, disconnects the PPMU, and closes the I/O switch connecting the instrument channel.

DISCONNECT
The I/O switch connecting the instrument channel is open to the I/O connector. If the PPMU is sourcing, it is stopped prior to opening the I/O switch.

SequencerFlag

`class nidigital.SequencerFlag`

FLAG0

FLAG1

FLAG2

FLAG3

SequencerRegister

`class nidigital.SequencerRegister`

REGISTER0

REGISTER1

REGISTER2

REGISTER3

REGISTER4

REGISTER5

REGISTER6

REGISTER7

REGISTER8

REGISTER9

REGISTER10

REGISTER11

REGISTER12

REGISTER13

REGISTER14

REGISTER15

SoftwareTrigger

`class nidigital.SoftwareTrigger`

START
Overrides the start trigger.

CONDITIONAL_JUMP
Specifies to route a pattern opcode event signal.

SourceDataMapping

class nidigital.**SourceDataMapping**

BROADCAST
Broadcasts the waveform you specify to all sites.

SITE_UNIQUE
Sources unique waveform data to each site.

TDREndpointTermination

class nidigital.**TDREndpointTermination**

OPEN
TDR channels are connected to an open circuit.

SHORT_TO_GROUND
TDR channels are connected to a short to ground.

TerminationMode

class nidigital.**TerminationMode**

ACTIVE_LOAD
The active load provides a constant current to a commutating voltage (Vcom).

VTERM
The pin driver drives Vterm.

HIGH_Z
The pin driver is in a non-drive state (in a high-impedance state) and the active load is disabled.

TimeSetEdgeType

class nidigital.**TimeSetEdgeType**

DRIVE_ON
Specifies the drive on edge of the time set.

DRIVE_DATA
Specifies the drive data edge of the time set.

DRIVE_RETURN
Specifies the drive return edge of the time set.

DRIVE_OFF
Specifies the drive off edge of the time set.

COMPARE_STROBE
Specifies the compare strobe of the time set.

DRIVE_DATA2
Specifies the drive data 2 edge of the time set.

DRIVE_RETURN2
Specifies the drive return 2 edge of the time set.

COMPARE_STROBE2
Specifies the compare strobe 2 of the time set.

TriggerType

class nidigital.**TriggerType**

NONE
Disables the start trigger.

DIGITAL_EDGE
Digital edge trigger.

SOFTWARE
Software start trigger.

WriteStaticPinState

class nidigital.**WriteStaticPinState**

ZERO
Specifies to drive low.

ONE
Specifies to drive high.

X
Specifies to not drive.

Exceptions and Warnings

Error

exception nidigital.errors.**Error**
Base exception type that all NI-Digital Pattern Driver exceptions derive from

DriverError

exception nidigital.errors.**DriverError**
An error originating from the NI-Digital Pattern Driver driver

UnsupportedConfigurationError

exception nidigital.errors.**UnsupportedConfigurationError**
An error due to using this module in an usupported platform.

DriverNotInstalledError

exception nidigital.errors.**DriverNotInstalledError**
An error due to using this module without the driver runtime installed.

InvalidRepeatedCapabilityError

exception nidigital.errors.**InvalidRepeatedCapabilityError**
An error due to an invalid character in a repeated capability

SelfTestError

exception nidigital.errors.**SelfTestError**
An error due to a failed self-test

DriverWarning

exception nidigital.errors.**DriverWarning**
A warning originating from the NI-Digital Pattern Driver driver

Examples

You can download all nidigital examples here

nidigital_burst_with_start_trigger.py

Listing 4: (nidigital_burst_with_start_trigger.py)

```
#!/usr/bin/python

import argparse
import nidigital
import os
import sys


def example(resource_name, options, trigger_source=None, trigger_edge=None):

    with nidigital.Session(resource_name=resource_name, options=options) as session:

        dir = os.path.join(os.path.dirname(__file__))

        # Load the pin map (.pinmap) created using the Digital Pattern Editor
```

(continues on next page)

(continued from previous page)

```python
        pin_map_filename = os.path.join(dir, 'PinMap.pinmap')
        session.load_pin_map(pin_map_filename)

        # Load the specifications (.specs), levels (.digilevels), and timing (.digitiming) sheets created using the Digital Pattern Editor
        spec_filename = os.path.join(dir, 'Specifications.specs')
        levels_filename = os.path.join(dir, 'PinLevels.digilevels')
        timing_filename = os.path.join(dir, 'Timing.digitiming')
        session.load_specifications_levels_and_timing(spec_filename, levels_filename, timing_filename)

        # Apply the settings from the levels and timing sheets we just loaded to the session
        session.apply_levels_and_timing(levels_filename, timing_filename)

        # Loading the pattern file (.digipat) created using the Digital Pattern Editor
        pattern_filename = os.path.join(dir, 'Pattern.digipat')
        session.load_pattern(pattern_filename)

        if trigger_source is None:
            print('Start bursting pattern')
        else:
            # Specify a source and edge for the external start trigger
            session.start_trigger_type = nidigital.TriggerType.DIGITAL_EDGE
            session.digital_edge_start_trigger_source = trigger_source
            session.digital_edge_start_trigger_edge = nidigital.DigitalEdge.RISING if trigger_edge == 'Rising' else nidigital.DigitalEdge.FALLING
            print('Wait for start trigger and then start bursting pattern')

        # If start trigger is configured, waiting for the trigger to start bursting and then blocks until the pattern is done bursting
        # Else just start bursting and block until the pattern is done bursting
        session.burst_pattern(start_label='new_pattern')

        # Disconnect all channels using programmable onboard switching
        session.selected_function = nidigital.SelectedFunction.DISCONNECT
    print('Done bursting pattern')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Demonstrates how to create and configure a session that bursts a pattern on the digital pattern instrument using a start trigger', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2,PXI1Slot3', help='Resource name of a NI digital pattern instrument. Ensure the resource name matches the instrument name in the pinmap file.')
    parser.add_argument('-s', '--simulate', default='True', choices=['True', 'False'], help='Whether to run on simulated hardware or real hardware')
    subparser = parser.add_subparsers(dest='command', help='Sub-command help')
    start_trigger = subparser.add_parser('start-trigger', help='Configure start trigger')
    start_trigger.add_argument('-ts', '--trigger-source', default='/PXI1Slot2/PXI_Trig0', help='Source terminal for the start trigger')
    start_trigger.add_argument('-te', '--trigger-edge', default='Rising', choices=['Rising', 'Falling'], help='Trigger on rising edge or falling edge of start trigger')
    args = parser.parse_args(argsv)
```

(continues on next page)

(continued from previous page)

```

    example(args.resource_name,
            'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
            args.trigger_source if args.command == 'start-trigger' else None,
            args.trigger_edge if args.command == 'start-trigger' else None)


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


def test_main():
    _main([])
    _main(['start-trigger'])


def test_example():
    resource_name = 'PXI1Slot2,PXI1Slot3'
    options = {'simulate': True, 'driver_setup': {'Model': '6571'}, }
    example(resource_name, options)

    trigger_source = '/PXI1Slot2/PXI_Trig0'
    trigger_edge = 'Rising'
    example(resource_name, options, trigger_source, trigger_edge)


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

nidigital_configure_time_set_and_voltage_levels.py

Listing 5: (nidigital_configure_time_set_and_voltage_levels.py)

```
#!/usr/bin/python

import argparse
import nidigital
import os
import sys


class VoltageLevelsAndTerminationConfig():
    def __init__(self, vil, vih, vol, voh, vterm, termination_mode, iol, ioh, vcom):
        self.vil = vil
        self.vih = vih
        self.vol = vol
        self.voh = voh
        self.vterm = vterm
        self.termination_mode = termination_mode
        self.iol = iol
        self.ioh = ioh
        self.vcom = vcom


```

(continues on next page)

(continued from previous page)

```
class TimeSetConfig():
    def __init__(self, time_set_name, period, drive_format, drive_on, drive_data,␣
↪drive_return, drive_off, strobe_edge):
        self.time_set_name = time_set_name
        self.period = period
        self.drive_format = drive_format
        self.drive_on = drive_on
        self.drive_data = drive_data
        self.drive_return = drive_return
        self.drive_off = drive_off
        self.strobe_edge = strobe_edge


def convert_drive_format(drive_format):
    converter = {'NR': nidigital.DriveFormat.NR,
                 'RL': nidigital.DriveFormat.RL,
                 'RH': nidigital.DriveFormat.RH,
                 'SBC': nidigital.DriveFormat.SBC}
    return converter.get(drive_format, None)


def example(resource_name,
            options,
            channels,
            voltage_config,
            time_set_config):

    with nidigital.Session(resource_name=resource_name, options=options) as session:

        dir = os.path.dirname(__file__)

        # Load pin map (.pinmap) created using Digital Pattern Editor
        pin_map_filename = os.path.join(dir, 'PinMap.pinmap')
        session.load_pin_map(pin_map_filename)

        # Configure voltage levels and terminal voltage through driver API
        session.channels[channels].configure_voltage_levels(voltage_config.vil,␣
↪voltage_config.vih, voltage_config.vol, voltage_config.voh, voltage_config.vterm)
        if voltage_config.termination_mode == 'High_Z':
            session.channels[channels].termination_mode = nidigital.TerminationMode.
↪HIGH_Z
        elif voltage_config.termination_mode == 'Active_Load':
            session.channels[channels].termination_mode = nidigital.TerminationMode.
↪ACTIVE_LOAD
            session.channels[channels].configure_active_load_levels(voltage_config.
↪iol, voltage_config.ioh, voltage_config.vcom)
        else:
            session.channels[channels].termination_mode = nidigital.TerminationMode.
↪VTERM

        # Configure time set through driver API
        session.create_time_set(time_set_config.time_set_name)  # Must match time set␣
↪name in pattern file
        session.configure_time_set_period(time_set_config.time_set_name, time_set_
↪config.period)
        session.channels[channels].configure_time_set_drive_edges(time_set_config.
↪time_set_name, convert_drive_format(time_set_config.drive_format),
```

(continues on next page)

(continued from previous page)

```
                                                                    time_set_config.
↪drive_on, time_set_config.drive_data,
                                                                    time_set_config.
↪drive_return, time_set_config.drive_off)
            session.channels[channels].configure_time_set_compare_edges_strobe(time_set_
↪config.time_set_name, time_set_config.strobe_edge)

        # Load the pattern file (.digipat) created using Digital Pattern Editor
        pattern_filename = os.path.join(dir, 'Pattern.digipat')
        session.load_pattern(pattern_filename)

        # Burst pattern, blocks until the pattern is done bursting
        session.burst_pattern(start_label='new_pattern')
        print('Start bursting pattern')

        # Disconnect all channels using programmable onboard switching
        session.selected_function = nidigital.SelectedFunction.DISCONNECT
    print('Done bursting pattern')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Demonstrates how to create an␣
↪instrument session, configure time set and voltage levels, and burst a pattern on␣
↪the digital pattern instrument.', formatter_class=argparse.
↪ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2,PXI1Slot3', help=
↪'Resource name of a NI digital pattern instrument, ensure the resource name matches␣
↪the instrument name in the pinmap file.')
    parser.add_argument('-s', '--simulate', default='True', choices=['True', 'False'],
↪ help='Whether to run on simulated hardware or on real hardware')
    parser.add_argument('-c', '--channels', default='PinGroup1', help='Channel(s)/
↪Pin(s) to configure')

    # Parameters to configure voltage
    parser.add_argument('--vil', default=0, type=float, help='The voltage that the␣
↪instrument will apply to the input of the DUT when the pin driver drives a logic␣
↪low (0)')
    parser.add_argument('--vih', default=3.3, type=float, help='The voltage that the␣
↪instrument will apply to the input of the DUT when the test instrument drives a␣
↪logic high (1)')
    parser.add_argument('--vol', default=1.6, type=float, help='The output voltage␣
↪below which the comparator on the pin driver interprets a logic low (L)')
    parser.add_argument('--voh', default=1.7, type=float, help='The output voltage␣
↪above which the comparator on the pin driver interprets a logic high (H)')
    parser.add_argument('--vterm', default=2, type=float, help='The termination␣
↪voltage the instrument applies during non-drive cycles when the termination mode is␣
↪set to Vterm')
    parser.add_argument('-term-mode', '--termination-mode', default='High_Z',␣
↪choices=['High_Z', 'Active_Load', 'Three_Level_Drive'])
    parser.add_argument('--iol', default=0.002, type=float, help='The maximum current␣
↪that the DUT sinks while outputting a voltage below VCOM')
    parser.add_argument('--ioh', default=-0.002, type=float, help='The maximum␣
↪current that the DUT sources while outputting a voltage above VCOM')
    parser.add_argument('--vcom', default=0.0, type=float, help='The commutating␣
↪voltage level at which the active load circuit switches between sourcing current␣
↪and sinking current')

```

(continues on next page)

(continued from previous page)

```
    # Parameters to configure timeset
    parser.add_argument('--period', default=0.00000002, type=float, help='Period in␣
↪second')
    parser.add_argument('-format', '--drive-format', default='NR', choices=['NR', 'RL
↪', 'RH', 'SBC'], help='Non-return | Return to low | Return to high | Surround by␣
↪complement')
    parser.add_argument('--drive-on', default=0, type=float, help='The delay in␣
↪seconds from the beginning of the vector period for turning on the pin driver')
    parser.add_argument('--drive-data', default=0, type=float, help='The delay in␣
↪seconds from the beginning of the vector period until the pattern data is driven to␣
↪the pattern value')
    parser.add_argument('--drive-return', default=0.000000015, type=float, help='The␣
↪delay in seconds from the beginning of the vector period until the pin changes from␣
↪the pattern data to the return value, as specified in the format.')
    parser.add_argument('--drive-off', default=0.00000002, type=float, help='The␣
↪delay in seconds from the beginning of the vector period to turn off the pin driver␣
↪when the next vector period uses a non-drive symbol (L, H, X, V, M, E).')
    parser.add_argument('--strobe-edge', default=0.00000001, type=float, help='The␣
↪time in second when the comparison happens within a vector period')

    args = parser.parse_args(argsv)
    voltage_config = VoltageLevelsAndTerminationConfig(args.vil, args.vih, args.vol,␣
↪args.voh, args.vterm, args.termination_mode, args.iol, args.ioh, args.vcom)
    time_set_config = TimeSetConfig("tset0", args.period, args.drive_format, args.
↪drive_on, args.drive_data, args.drive_return, args.drive_off, args.strobe_edge)
    example(args.resource_name,
            'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
            args.channels,
            voltage_config,
            time_set_config)


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


def test_main():
    _main([])


def test_example():
    resource_name = 'PXI1Slot2,PXI1Slot3'
    options = {'simulate': True, 'driver_setup': {'Model': '6571'}, }
    channels = 'PinGroup1'
    voltage_config = VoltageLevelsAndTerminationConfig(vil=0, vih=3.3, vol=1.6, voh=1.
↪7, vterm=2,
                                                        termination_mode='Active_Load',
↪ iol=0.002, ioh=-0.002, vcom=0)
    time_set_config = TimeSetConfig(time_set_name="tset0",
                                    period=0.00000002,
                                    drive_format='NR',
                                    drive_on=0, drive_data=0, drive_return=0.
↪000000015, drive_off=0.00000002, strobe_edge=0.00000001)
    example(resource_name, options, channels, voltage_config, time_set_config)


if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    main()
```

nidigital_ppmu_source_and_measure.py

Listing 6: (nidigital_ppmu_source_and_measure.py)

```
#!/usr/bin/python

import argparse
import nidigital
import os
import pytest
import sys
import time


def example(resource_name, options, channels, measure, aperture_time,
            source=None, settling_time=None, current_level_range=None, current_
↪level=None,
            voltage_limit_high=None, voltage_limit_low=None, current_limit_range=None,
↪ voltage_level=None):

    with nidigital.Session(resource_name=resource_name, options=options) as session:

        dir = os.path.join(os.path.dirname(__file__))

        # Load pin map (.pinmap) created using Digital Pattern Editor
        pin_map_filename = os.path.join(dir, 'PinMap.pinmap')
        session.load_pin_map(pin_map_filename)

        # Configure the PPMU measurement aperture time
        session.channels[channels].ppmu_aperture_time = aperture_time
        session.channels[channels].ppmu_aperture_time_units = nidigital.
↪PPMUApertureTimeUnits.SECONDS

        # Configure and source
        if source == 'source-current':
            session.channels[channels].ppmu_output_function = nidigital.
↪PPMUOutputFunction.CURRENT

            session.channels[channels].ppmu_current_level_range = current_level_range
            session.channels[channels].ppmu_current_level = current_level
            session.channels[channels].ppmu_voltage_limit_high = voltage_limit_high
            session.channels[channels].ppmu_voltage_limit_low = voltage_limit_low

            session.channels[channels].ppmu_source()

            # Settling time between sourcing and measuring
            time.sleep(settling_time)

        elif source == 'source-voltage':
            session.channels[channels].ppmu_output_function = nidigital.
↪PPMUOutputFunction.VOLTAGE

```

(continues on next page)

(continued from previous page)

```
            session.channels[channels].ppmu_current_limit_range = current_limit_range
            session.channels[channels].ppmu_voltage_level = voltage_level

            session.channels[channels].ppmu_source()

            # Settling time between sourcing and measuring
            time.sleep(settling_time)

        pin_info = session.channels[channels].get_pin_results_pin_information()

        # Measure
        if measure == 'current':
            current_measurements = session.channels[channels].ppmu_measure(nidigital.
↪PPMUMeasurementType.CURRENT)

            print('{:<6} {:<20} {:<10}'.format('Site', 'Pin Name', 'Current'))

            for pin, current in zip(pin_info, current_measurements):
                print('{:<6d} {:<20} {:<10f}'.format(pin.site_number, pin.pin_name,␣
↪current))
        else:
            voltage_measurements = session.channels[channels].ppmu_measure(nidigital.
↪PPMUMeasurementType.VOLTAGE)

            print('{:<6} {:<20} {:<10}'.format('Site', 'Pin Name', 'Voltage'))

            for pin, voltage in zip(pin_info, voltage_measurements):
                print('{:<6d} {:<20} {:<10f}'.format(pin.site_number, pin.pin_name,␣
↪voltage))

        # Disconnect all channels using programmable onboard switching
        session.channels[channels].selected_function = nidigital.SelectedFunction.
↪DISCONNECT


def _main(argsv):
    parser = argparse.ArgumentParser(description='Demonstrates how to source/measure␣
↪voltage/current using the PPMU on selected channels/pins of the digital pattern␣
↪instrument',
                                     formatter_class=argparse.
↪ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2,PXI1Slot3', help=
↪'Resource name of a NI digital pattern instrument, ensure the resource name matches␣
↪the instrument name in the pinmap file.')
    parser.add_argument('-s', '--simulate', default='True', choices=['True', 'False'],
↪ help='Whether to run on simulated hardware or on real hardware')
    parser.add_argument('-c', '--channels', default='DUTPin1, SystemPin1', help=
↪'Channel(s)/Pin(s) to use')
    parser.add_argument('-m', '--measure', default='voltage', choices=['voltage',
↪'current'], help='Measure voltage or measure current')
    parser.add_argument('-at', '--aperture-time', default=0.000004, type=float, help=
↪'Aperture time in seconds')
    subparser = parser.add_subparsers(dest='source', help='Sub-command help, by␣
↪default it measures voltage and does not source')

    source_current = subparser.add_parser('source-current', help='Source current')
    source_current.add_argument('-clr', '--current-level-range', default=0.000002,␣
↪type=float, help='Current level range in amps')
```

(continues on next page)

(continued from previous page)

```python
    source_current.add_argument('-cl', '--current-level', default=0.000002, type=float, help='Current level in amps')
    source_current.add_argument('-vlh', '--voltage-limit-high', default=3.3, type=float, help='Voltage limit high in volts')
    source_current.add_argument('-vll', '--voltage-limit-low', default=0, type=float, help='Voltage limit low in volts')
    source_current.add_argument('-st', '--settling-time', default=0.01, type=float, help='Settling time in seconds')

    source_voltage = subparser.add_parser('source-voltage', help='Source voltage')
    source_voltage.add_argument('-clr', '--current-limit-range', default=0.000002, type=float, help='Current limit range in amps')
    source_voltage.add_argument('-vl', '--voltage-level', default=3.3, type=float, help='Voltage level in volts')
    source_voltage.add_argument('-st', '--settling-time', default=0.01, type=float, help='Settling time in seconds')

    args = parser.parse_args(argsv)

    if args.source == 'source-current':
        example(
            args.resource_name,
            'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
            args.channels,
            args.measure,
            args.aperture_time,
            args.source,
            args.settling_time,
            args.current_level_range,
            args.current_level,
            args.voltage_limit_high,
            args.voltage_limit_low)
    elif args.source == 'source-voltage':
        example(
            args.resource_name,
            'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
            args.channels,
            args.measure,
            args.aperture_time,
            args.source,
            args.settling_time,
            current_limit_range=args.current_limit_range,
            voltage_level=args.voltage_level)
    else:
        if args.measure == 'current':
            raise ValueError('Cannot measure current on a channel that is not sourcing voltage or current')
        example(
            args.resource_name,
            'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
            args.channels,
            args.measure,
            args.aperture_time)


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

(continues on next page)

(continued from previous page)

```python


def test_main():
    _main([])
    _main(['-m', 'voltage'])
    with pytest.raises(Exception):
        _main(['-m', 'current'])
    _main(['-m', 'voltage', 'source-current'])
    _main(['-m', 'current', 'source-current'])
    _main(['-m', 'voltage', 'source-voltage'])
    _main(['-m', 'current', 'source-voltage'])


def test_example():
    resource_name = 'PXI1Slot2,PXI1Slot3'
    options = {'simulate': True, 'driver_setup': {'Model': '6571'}, }
    channels = 'DUTPin1, SystemPin1'
    aperture_time = 0.000004

    example(resource_name, options, channels, 'voltage',
            aperture_time)
    with pytest.raises(Exception):
        example(resource_name, options, channels, 'current',
                aperture_time)

    settling_time = 0.01
    current_level_range = 0.000002
    current_level = 0.000002
    voltage_limit_high = 3.3
    voltage_limit_low = 0
    example(resource_name, options, channels, 'voltage',
            aperture_time, 'source-current', settling_time,
            current_level_range, current_level,
            voltage_limit_high, voltage_limit_low)
    example(resource_name, options, channels, 'current',
            aperture_time, 'source-current', settling_time,
            current_level_range, current_level,
            voltage_limit_high, voltage_limit_low)

    current_limit_range = 0.000002
    voltage_level = 3.3
    example(resource_name, options, channels, 'voltage',
            aperture_time, 'source-voltage', settling_time,
            current_limit_range=current_limit_range,
            voltage_level=voltage_level)
    example(resource_name, options, channels, 'current',
            aperture_time, 'source-voltage', settling_time,
            current_limit_range=current_limit_range,
            voltage_level=voltage_level)


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

7.3 nidmm module

7.3.1 Installation

As a prerequisite to using the nidmm module, you must install the NI-DMM runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

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

```
$ python -m pip install nidmm~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install nidmm
```

7.3.2 Usage

The following is a basic example of using the **nidmm** module to open a session to a DMM and perform a 5.5 digits of resolution voltage measurement in the 10 V range.

```python
import nidmm
with nidmm.Session("Dev1") as session:
    session.configureMeasurementDigits(nidmm.Function.DC_VOLTS, 10, 5.5)
    print("Measurement: " + str(session.read()))
```

Additional examples for NI-DMM are located in src/nidmm/examples/ directory.

7.3.3 API Reference

Session

class nidmm.**Session**(*self*, *resource_name*, *id_query=False*, *reset_device=False*, *options={}*)

This method completes the following tasks:

- Creates a new IVI instrument driver session and, optionally, sets the initial state of the following session properties: `nidmm.Session.RANGE_CHECK`, `nidmm.Session.QUERY_INSTR_STATUS`, `nidmm.Session.CACHE`, *`nidmm.Session.simulate`*, `nidmm.Session.RECORD_COERCIONS`.
- Opens a session to the device you specify for the **Resource_Name** parameter. If the **ID_Query** parameter is set to True, this method queries the instrument ID and checks that it is valid for this instrument driver.
- If the **Reset_Device** parameter is set to True, this method resets the instrument to a known state. Sends initialization commands to set the instrument to the state necessary for the operation of the instrument driver.
- Returns a ViSession handle that you use to identify the instrument in all subsequent instrument driver method calls.

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

Parameters

- **resource_name** (*str*) –

> **Caution:** All IVI names for the **Resource_Name**, such as logical names or virtual 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 in the name.

  Contains the **resource_name** of the device to initialize. The **resource_name** is assigned in Measurement & Automation Explorer (MAX). Refer to Related Documentation for the *NI Digital Multimeters Getting Started Guide* for more information about configuring and testing the DMM in MAX.
  Valid Syntax:

  – NI-DAQmx name

  – DAQ::NI-DAQmx name[::INSTR]

  – DAQ::Traditional NI-DAQ device number[::INSTR]

  – IVI logical name

- **id_query** (*bool*) – Verifies that the device you initialize is one that the driver supports. NI-DMM automatically performs this query, so setting this parameter is not necessary. Defined Values:

True (default)	1	Perform ID Query
False	0	Skip ID Query

- **reset_device** (*bool*) – Specifies whether to reset the instrument during the initialization procedure. Defined Values:

True (default)	1	Reset Device
False	0	Don't Reset

- **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	{}

Methods

abort

`nidmm.Session.`**`abort`**`()`

Aborts a previously initiated measurement and returns the DMM to the Idle state.

close

`nidmm.Session.`**`close`**`()`

Closes the specified session and deallocates resources that it reserved.

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

configure_measurement_absolute

`nidmm.Session.`**`configure_measurement_absolute`**`(`*measurement_function*, *range*, *resolution_absolute*`)`

Configures the common properties of the measurement. These properties include `nidmm.Session.method`, *`nidmm.Session.range`*, and *`nidmm.Session.resolution_absolute`*.

Parameters

- **`measurement_function`** (*`nidmm.Function`*) – Specifies the **measurement_function** used to acquire the measurement. The driver sets `nidmm.Session.method` to this value.
- **`range`** (*`float`*) – Specifies the **range** for the method specified in the **Measurement_Function** parameter. When frequency is specified in the **Measurement_Function** parameter, you must supply the minimum frequency expected in the **range** parameter. For example, you must type in 100 Hz if you are measuring 101 Hz or higher. For all other methods, you must supply a **range** that exceeds the value that you are measuring. For example, you must type in 10 V if you are measuring 9 V. **range** values are coerced up to the closest input **range**. Refer to the Devices Overview for a list of valid ranges. The driver sets *`nidmm.Session.range`* to this value. The default is 0.02 V.

`NIDMM_VAL_AUTO_RANGE_ON`	-1.0	NI-DMM performs an Auto Range before acquiring the measurement.
`NIDMM_VAL_AUTO_RANGE_OFF`	-2.0	NI-DMM sets the Range to the current *`nidmm.Session.auto_range_value`* and uses this range for all subsequent measurements until the measurement configuration is changed.
`NIDMM_VAL_AUTO_RANGE_ONCE`	-3.0	NI-DMM performs an Auto Range before acquiring the measurement. The *`nidmm.Session.auto_range_value`* is stored and used for all subsequent measurements until the measurement configuration is changed.

Note: The NI 4050, NI 4060, and NI 4065 only support Auto Range when the trigger and sample trigger are set to IMMEDIATE.

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.

- **`resolution_absolute`** (*`float`*) – Specifies the absolute resolution for the measurement. NI-DMM sets *`nidmm.Session.resolution_absolute`* to this value. The PXIe-4080/4081/4082 uses the resolution you specify. The NI 4065 and NI 4070/4071/4072 ignore this parameter when the **Range** parameter is set to `NIDMM_VAL_AUTO_RANGE_ON` (-1.0) or `NIDMM_VAL_AUTO_RANGE_ONCE` (-3.0). The default is 0.001 V.

  Note: NI-DMM ignores this parameter for capacitance and inductance measurements on the NI 4072. To achieve better resolution for such measurements, use the *`nidmm.Session.lc_number_meas_to_average`* property.

  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.

configure_measurement_digits

`nidmm.Session.`**`configure_measurement_digits`**(*measurement_function*, *range*, *resolution_digits*)

Configures the common properties of the measurement. These properties include `nidmm.Session.method`, *`nidmm.Session.range`*, and *`nidmm.Session.resolution_digits`*.

Parameters

- **`measurement_function`** (*`nidmm.Function`*) – Specifies the **measurement_function** used to acquire the measurement. The driver sets `nidmm.Session.method` to this value.
- **`range`** (*`float`*) – Specifies the range for the method specified in the **Measurement_Function** parameter. When frequency is specified in the **Measurement_Function** parameter, you must supply the minimum frequency expected in the **range** parameter. For example, you must type in 100 Hz if you are measuring 101 Hz or higher. For all other methods, you must supply a range that exceeds the value that you are measuring. For example, you must type in 10 V if you are measuring 9 V. range values are coerced up to the closest input range. Refer to the Devices Overview for a list of valid ranges. The driver sets *`nidmm.Session.range`* to this value. The default is 0.02 V.

NIDMM_VAL_AUTO_RANGE_ON	-1.0	NI-DMM performs an Auto Range before acquiring the measurement.
NIDMM_VAL_AUTO_RANGE_OFF	-2.0	NI-DMM sets the Range to the current *nidmm.Session.auto_range_value* and uses this range for all subsequent measurements until the measurement configuration is changed.
NIDMM_VAL_AUTO_RANGE_ONCE	-3.0	NI-DMM performs an Auto Range before acquiring the measurement. The *nidmm.Session.auto_range_value* is stored and used for all subsequent measurements until the measurement configuration is changed.

Note: The NI 4050, NI 4060, and NI 4065 only support Auto Range when the trigger and sample trigger are set to IMMEDIATE.

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.

- **resolution_digits** (*float*) – Specifies the resolution of the measurement in digits. The driver sets the Devices Overview for a list of valid ranges. The driver sets *nidmm.Session.resolution_digits* property to this value. The PXIe-4080/4081/4082 uses the resolution you specify. The NI 4065 and NI 4070/4071/4072 ignore this parameter when the **Range** parameter is set to NIDMM_VAL_AUTO_RANGE_ON (-1.0) or NIDMM_VAL_AUTO_RANGE_ONCE (-3.0). The default is 5½.

Note: NI-DMM ignores this parameter for capacitance and inductance measurements on the NI 4072. To achieve better resolution for such measurements, use the *nidmm.Session.lc_number_meas_to_average* property.

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.

configure_multi_point

nidmm.Session.**configure_multi_point**(*trigger_count*, *sample_count*, *sample_trigger=nidmm.SampleTrigger.IMMEDIATE*, *sample_interval=hightime.timedelta(seconds=-1)*)

Configures the properties for multipoint measurements. These properties include *nidmm.Session.trigger_count*, *nidmm.Session.sample_count*, *nidmm.Session.sample_trigger*, and *nidmm.Session.sample_interval*.

For continuous acquisitions, set *nidmm.Session.trigger_count* or *nidmm.Session.sample_count* to zero. For more information, refer to Multiple Point Acquisitions, Triggering, and Using Switches.

Parameters

- **trigger_count** (*int*) – Sets the number of triggers you want the DMM to receive before returning to the Idle state. The driver sets `nidmm.Session.trigger_count` to this value. The default value is 1.
- **sample_count** (*int*) – Sets the number of measurements the DMM makes in each measurement sequence initiated by a trigger. The driver sets `nidmm.Session.sample_count` to this value. The default value is 1.
- **sample_trigger** (`nidmm.SampleTrigger`) – Specifies the **sample_trigger** source you want to use. The driver sets `nidmm.Session.sample_trigger` to this value. The default is Immediate.

  Note: To determine which values are supported by each device, refer to the LabWindows/CVI Trigger Routing section.

- **sample_interval** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Sets the amount of time in seconds the DMM waits between measurement cycles. The driver sets `nidmm.Session.sample_interval` to this value. Specify a sample interval to add settling time between measurement cycles or to decrease the measurement rate. **sample_interval** only applies when the **Sample_Trigger** is set to INTERVAL.

  On the NI 4060, the **sample_interval** value is used as the settling time. When sample interval is set to 0, the DMM does not settle between measurement cycles. The NI 4065 and NI 4070/4071/4072 use the value specified in **sample_interval** as additional delay. The default value (-1) ensures that the DMM settles for a recommended time. This is the same as using an Immediate trigger.

  Note: This property is not used on the NI 4080/4081/4082 and the NI 4050.

configure_rtd_custom

`nidmm.Session.`**`configure_rtd_custom`**`(`*rtd_a*, *rtd_b*, *rtd_c*`)`

Configures the A, B, and C parameters for a custom RTD.

Parameters

- **rtd_a** (*float*) – Specifies the Callendar-Van Dusen A coefficient for RTD scaling when RTD Type parameter is set to Custom in the `nidmm.Session.configure_rtd_type()` method. The default is 3.9083e-3 (Pt3851)
- **rtd_b** (*float*) – Specifies the Callendar-Van Dusen B coefficient for RTD scaling when RTD Type parameter is set to Custom in the `nidmm.Session.configure_rtd_type()` method. The default is -5.775e-7 (Pt3851).
- **rtd_c** (*float*) – Specifies the Callendar-Van Dusen C coefficient for RTD scaling when RTD Type parameter is set to Custom in the `nidmm.Session.configure_rtd_type()` method. The default is -4.183e-12 (Pt3851).

configure_rtd_type

nidmm.Session.**configure_rtd_type**(*rtd_type*, *rtd_resistance*)

Configures the RTD Type and RTD Resistance parameters for an RTD.

Parameters

- **rtd_type** (*nidmm.RTDType*) – Specifies the type of RTD used to measure the temperature resistance. NI-DMM uses this value to set the RTD Type property. The default is *PT3851*.

Enum	Standards	Material	TCR (α)	Typical R_0 (Ω)	Notes	
Callendar-Van Dusen Coefficient						
PT3851	IEC-751 DIN 43760 BS 1904 ASTM-E1137 EN-60751	Platinum	.003851	100 Ω 1000 Ω	A = 3.9083 × 10^{-3} B = –5.775×10:sup:*–7* C = –4.183×10:sup:*–12*	Most common RTDs
PT3750	Low-cost vendor compliant RTD*	Platinum	.003750	1000 Ω	A = 3.81 × 10^{-3} B = –6.02×10:sup:*–7* C = –6.0×10:sup:*–12*	Low-cost RTD
PT3916	JISC 1604	Platinum	.003916	100 Ω	A = 3.9739 × 10^{-3} B = –5.870×10:sup:*–7* C = –4.4 ×10^{-12}	Used in primarily in Japan
PT3920	US Industrial Standard D-100 American	Platinum	.003920	100 Ω	A = 3.9787 × 10^{-3} B = –5.8686×10:sup:*–7* C = –4.167 ×10^{-12}	Low-cost RTD
PT3911	US Industrial Standard American	Platinum	.00391	1100 Ω	A = 3.9692 × 10^{-3} B = –5.8495×10:sup:*–7* C = –4.233 ×10^{-12}	Low-cost RTD
PT3928	ITS-90	Platinum	.003928	100 Ω	A = 3.9888 × 10^{-3} B = –5.915×10:sup:*–7* C = –3.85 ×10^{-12}	The definition of temperature
*No standard. Check the TCR.						

- **rtd_resistance** (*float*) – Specifies the RTD resistance in ohms at 0 °C. NI-DMM uses this value to set the RTD Resistance property. The default is 100 (Ω).

configure_thermistor_custom

`nidmm.Session.`**`configure_thermistor_custom`**(*thermistor_a*, *thermistor_b*, *thermistor_c*)

Configures the A, B, and C parameters for a custom thermistor.

Parameters

- **`thermistor_a`** (*`float`*) – Specifies the Steinhart-Hart A coefficient for thermistor scaling when Thermistor Type is set to Custom in the `nidmm.Session.ConfigureThermistorType()` method. The default is 1.0295e-3 (44006).

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

- **`thermistor_b`** (*`float`*) – Specifies the Steinhart-Hart B coefficient for thermistor scaling when Thermistor Type is set to Custom in the `nidmm.Session.ConfigureThermistorType()` method. The default is 2.391e-4 (44006).

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

- **`thermistor_c`** (*`float`*) – Specifies the Steinhart-Hart C coefficient for thermistor scaling when Thermistor Type is set to Custom in the `nidmm.Session.ConfigureThermistorType()` method. The default is 1.568e-7 (44006).

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

configure_thermocouple

`nidmm.Session.`**`configure_thermocouple`**(*thermocouple_type*, *reference_junction_type=nidmm.ThermocoupleReferenceJunctionType.FIXED*

Configures the thermocouple type and reference junction type for a chosen thermocouple.

Parameters

- **`thermocouple_type`** (*`nidmm.ThermocoupleType`*) – Specifies the type of thermocouple used to measure the temperature. NI-DMM uses this value to set the Thermocouple Type property. The default is *`J`*.

B	Thermocouple type B
E	Thermocouple type E
J	Thermocouple type J
K	Thermocouple type K
N	Thermocouple type N
R	Thermocouple type R
S	Thermocouple type S
T	Thermocouple type T

- **reference_junction_type** (*nidmm.ThermocoupleReferenceJunctionType*) – Specifies the type of reference junction to be used in the reference junction compensation of a thermocouple measurement. NI-DMM uses this value to set the Reference Junction Type property. The only supported value is `NIDMM_VAL_TEMP_REF_JUNC_FIXED`.

  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.

configure_trigger

`nidmm.Session.`**`configure_trigger`**(*trigger_source*, *trigger_delay=hightime.timedelta(seconds=-1)*)

Configures the DMM **Trigger_Source** and **Trigger_Delay**. Refer to Triggering and Using Switches for more information.

Parameters

- **trigger_source** (*nidmm.TriggerSource*) – Specifies the **trigger_source** that initiates the acquisition. The driver sets *nidmm.Session.trigger_source* to this value. Software configures the DMM to wait until *nidmm.Session.send_software_trigger()* is called before triggering the DMM.

  Note: To determine which values are supported by each device, refer to the LabWindows/CVI Trigger Routing section.

- **trigger_delay** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Specifies the time that the DMM waits after it has received a trigger before taking a measurement. The driver sets the *nidmm.Session.trigger_delay* property to this value. By default, **trigger_delay** is `NIDMM_VAL_AUTO_DELAY` (-1), which means the DMM waits an appropriate settling time before taking the measurement. On the NI 4060, if you set **trigger_delay** to 0, the DMM does not settle before taking the measurement. The NI 4065 and NI 4070/4071/4072 use the value specified in **trigger_delay** as additional settling time.

  Note: When using the NI 4050, **Trigger_Delay** must be set to `NIDMM_VAL_AUTO_DELAY` (-1).

  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.

configure_waveform_acquisition

`nidmm.Session.`**`configure_waveform_acquisition`**(*measurement_function*, *range*, *rate*, *waveform_points*)

Configures the DMM for waveform acquisitions. This feature is supported on the NI

4080/4081/4082 and the NI 4070/4071/4072.

Parameters

- **measurement_function** (*nidmm.Function*) – Specifies the **measurement_function** used in a waveform acquisition. The driver sets nidmm.Session.method to this value.

WAVEFORM_VOLTAGE (default)	1003	Voltage Waveform
WAVEFORM_CURRENT	1004	Current Waveform

- **range** (*float*) – Specifies the expected maximum amplitude of the input signal and sets the **range** for the **Measurement_Function**. NI-DMM sets *nidmm.Session.range* to this value. **range** values are coerced up to the closest input **range**. The default is 10.0.

  For valid ranges refer to the topics in Devices.

  Auto-ranging is not supported during waveform acquisitions.

- **rate** (*float*) – Specifies the **rate** of the acquisition in samples per second. NI-DMM sets *nidmm.Session.waveform_rate* to this value.

  The valid **Range** is 10.0–1,800,000 S/s. **rate** values are coerced to the closest integer divisor of 1,800,000. The default value is 1,800,000.

- **waveform_points** (*int*) – Specifies the number of points to acquire before the waveform acquisition completes. NI-DMM sets *nidmm.Session.waveform_points* to this value.

  To calculate the maximum and minimum number of waveform points that you can acquire in one acquisition, refer to the Waveform Acquisition Measurement Cycle.

  The default value is 500.

disable

nidmm.Session.**disable**()

Places the instrument in a quiescent state where it has minimal or no impact on the system to which it is connected. If a measurement is in progress when this method is called, the measurement is aborted.

export_attribute_configuration_buffer

nidmm.Session.**export_attribute_configuration_buffer**()

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

You can export and import session property configurations only between devices with identical model numbers.

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

Coercion Behavior for Certain Devices

Imported and exported property configurations contain coerced values for the following NI-DMM devices:

- PXI/PCI/PCIe/USB-4065
- PXI/PCI-4070
- PXI-4071
- PXI-4072

NI-DMM coerces property values when the value you set is within the allowed range for the property but is not one of the discrete valid values the property supports. For example, for a property that coerces values up, if you choose a value of 4 when the adjacent valid values are 1 and 10, the property coerces the value to 10.

Related Topics:

Using Properties and Properties with NI-DMM

Setting Properties Before Reading Properties

Note: Not supported on the PCMCIA-4050 or the PXI/PCI-4060.

Return type bytes

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

export_attribute_configuration_file

`nidmm.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.

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

Coercion Behavior for Certain Devices

Imported and exported property configurations contain coerced values for the following NI-DMM devices:

- PXI/PCI/PCIe/USB-4065
- PXI/PCI-4070
- PXI-4071
- PXI-4072

NI-DMM coerces property values when the value you set is within the allowed range for the property but is not one of the discrete valid values the property supports. For example, for a property that coerces values up, if you choose a value of 4 when the adjacent valid values are 1 and 10, the property coerces the value to 10.

Related Topics:

Using Properties and Properties with NI-DMM

Setting Properties Before Reading Properties

Note: Not supported on the PCMCIA-4050 or the PXI/PCI-4060.

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:** .nidmmconfig

fetch

`nidmm.Session.`**`fetch`**(*maximum_time=hightime.timedelta(milliseconds=-1)*)

Returns the value from a previously initiated measurement. You must call `nidmm.Session._initiate()` before calling this method.

Parameters **`maximum_time`** (*`hightime.timedelta, datetime.timedelta, or int in milliseconds`*) – Specifies the **maximum_time** allowed for this method to complete in milliseconds. If the method does not complete within this time interval, the method returns the NIDMM_ERROR_MAX_TIME_EXCEEDED error code. This may happen if an external trigger has not been received, or if the specified timeout is not long enough for the acquisition to complete.

The valid range is 0–86400000. The default value is `NIDMM_VAL_TIME_LIMIT_AUTO` (-1). The DMM calculates the timeout automatically.

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.

Return type float

Returns The measured value returned from the DMM.

fetch_multi_point

`nidmm.Session.`**`fetch_multi_point`**(*array_size, maximum_time=hightime.timedelta(milliseconds=-1)*)

Returns an array of values from a previously initiated multipoint measurement. The number of measurements the DMM makes is determined by the values you specify for the **Trigger_Count** and **Sample_Count** parameters of *`nidmm.Session.configure_multi_point()`*. You must first call `nidmm.Session._initiate()` to initiate a measurement before calling this method.

Parameters

- **`array_size`** (*`int`*) – Specifies the number of measurements to acquire. The maximum number of measurements for a finite acquisition is the (**Trigger Count** x **Sample Count**) parameters in *`nidmm.Session.configure_multi_point()`*.

  For continuous acquisitions, up to 100,000 points can be returned at once. The number of measurements can be a subset. The valid range is any positive ViInt32. The default value is 1.

- **`maximum_time`** (*`hightime.timedelta, datetime.timedelta, or int in milliseconds`*) – Specifies the **maximum_time** allowed for this method to complete in milliseconds. If the method does not complete within this time interval, the method returns the NIDMM_ERROR_MAX_TIME_EXCEEDED error code. This may happen if an external trigger has not been received, or if the specified timeout is not long enough for the acquisition to complete.

The valid range is 0–86400000. The default value is `NIDMM_VAL_TIME_LIMIT_AUTO` (-1). The DMM calculates the timeout automatically.

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.

Return type

tuple (reading_array, actual_number_of_points)

WHERE

reading_array (array.array("d")):

An array of measurement values.

Note: The size of the **Reading_Array** must be at least the size that you specify for the **Array_Size** parameter.

actual_number_of_points (int):

Indicates the number of measured values actually retrieved from the DMM.

fetch_waveform

`nidmm.Session.`**`fetch_waveform`**(*array_size*, *maximum_time=hightime.timedelta(milliseconds=-1)*)

For the NI 4080/4081/4082 and the NI 4070/4071/4072, returns an array of values from a previously initiated waveform acquisition. You must call `nidmm.Session._initiate()` before calling this method.

Parameters

- **`array_size`** (*`int`*) – Specifies the number of waveform points to return. You specify the total number of points that the DMM acquires in the **Waveform Points** parameter of *`nidmm.Session.configure_waveform_acquisition()`*. The default value is 1.
- **`maximum_time`** (*`hightime.timedelta, datetime.timedelta, or int in milliseconds`*) – Specifies the **maximum_time** allowed for this method to complete in milliseconds. If the method does not complete within this time interval, the method returns the NIDMM_ERROR_MAX_TIME_EXCEEDED error code. This may happen if an external trigger has not been received, or if the specified timeout is not long enough for the acquisition to complete.

  The valid range is 0–86400000. The default value is `NIDMM_VAL_TIME_LIMIT_AUTO` (-1). The DMM calculates the timeout automatically.

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.

Return type

tuple (waveform_array, actual_number_of_points)

WHERE

waveform_array (array.array("d")):

Waveform Array is an array of measurement values stored in waveform data type.

actual_number_of_points (int):

Indicates the number of measured values actually retrieved from the DMM.

fetch_waveform_into

`nidmm.Session.`**`fetch_waveform_into`**(*array_size*, *maximum_time=hightime.timedelta(milliseconds=-1)*)

For the NI 4080/4081/4082 and the NI 4070/4071/4072, returns an array of values from a previously initiated waveform acquisition. You must call `nidmm.Session._initiate()` before calling this method.

Parameters

- **`waveform_array`** (*`numpy.array(dtype=numpy.float64)`*) – **Waveform Array** is an array of measurement values stored in waveform data type.
- **`maximum_time`** (*`hightime.timedelta, datetime.timedelta, or int in milliseconds`*) – Specifies the **maximum_time** allowed for this method to complete in milliseconds. If the method does not complete within this time interval, the method returns the NIDMM_ERROR_MAX_TIME_EXCEEDED error code. This may happen if an external trigger has not been received, or if the specified timeout is not long enough for the acquisition to complete.

  The valid range is 0–86400000. The default value is `NIDMM_VAL_TIME_LIMIT_AUTO` (-1). The DMM calculates the timeout automatically.

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.

Return type

tuple (waveform_array, actual_number_of_points)

WHERE

waveform_array (numpy.array(dtype=numpy.float64)):

Waveform Array is an array of measurement values stored in waveform data type.

actual_number_of_points (int):

Indicates the number of measured values actually retrieved from the DMM.

get_cal_date_and_time

`nidmm.Session.`**`get_cal_date_and_time`**`(`*cal_type*`)`

Returns the date and time of the last calibration performed.

Note: The NI 4050 and NI 4060 are not supported.

Parameters **`cal_type`** (*`int`*) – Specifies the type of calibration performed (external or self-calibration).

`NIDMM_VAL_INTERNAL_AREA` (default)	0	Self-Calibration
`NIDMM_VAL_EXTERNAL_AREA`	1	External Calibration

Note: The NI 4065 does not support self-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.

Return type hightime.datetime

Returns Indicates date and time of the last calibration.

get_dev_temp

`nidmm.Session.`**`get_dev_temp`**`(`*options=""*`)`

Returns the current **Temperature** of the device.

Note: The NI 4050 and NI 4060 are not supported.

Parameters **`options`** (*`str`*) – Reserved.

Return type float

Returns Returns the current **temperature** of the device.

get_ext_cal_recommended_interval

`nidmm.Session.`**`get_ext_cal_recommended_interval`**`()`

Returns the recommended interval between external recalibration in **Months**.

Note: The NI 4050 and NI 4060 are not supported.

Return type hightime.timedelta

Returns Returns the recommended number of **months** between external calibrations.

get_last_cal_temp

`nidmm.Session.`**`get_last_cal_temp`**`(`*cal_type*`)`

Returns the **Temperature** during the last calibration procedure.

Note: The NI 4050 and NI 4060 are not supported.

Parameters **`cal_type`** (*`int`*) – Specifies the type of calibration performed (external or self-calibration).

`NIDMM_VAL_INTERNAL_AREA` (default)	0	Self-Calibration
`NIDMM_VAL_EXTERNAL_AREA`	1	External Calibration

Note: The NI 4065 does not support self-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.

Return type float

Returns Returns the **temperature** during the last calibration.

get_self_cal_supported

`nidmm.Session.`**`get_self_cal_supported`**`()`

Returns a Boolean value that expresses whether or not the DMM that you are using can perform self-calibration.

Return type bool

Returns

Returns whether Self Cal is supported for the device specified by the given session.

True	1	The DMM that you are using can perform self-calibration.
False	0	The DMM that you are using cannot perform self-calibration.

import_attribute_configuration_buffer

`nidmm.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.

Coercion Behavior for Certain Devices

Imported and exported property configurations contain coerced values for the following NI-DMM devices:

- PXI/PCI/PCIe/USB-4065
- PXI/PCI-4070
- PXI-4071
- PXI-4072

NI-DMM coerces property values when the value you set is within the allowed range for the property but is not one of the discrete valid values the property supports. For example, for a property that coerces values up, if you choose a value of 4 when the adjacent valid values are 1 and 10, the property coerces the value to 10.

Related Topics:

Using Properties and Properties with NI-DMM

Setting Properties Before Reading Properties

Note: Not supported on the PCMCIA-4050 or the PXI/PCI-4060.

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

import_attribute_configuration_file

`nidmm.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.

Coercion Behavior for Certain Devices

Imported and exported property configurations contain coerced values for the following NI-DMM devices:

- PXI/PCI/PCIe/USB-4065
- PXI/PCI-4070
- PXI-4071
- PXI-4072

NI-DMM coerces property values when the value you set is within the allowed range for the property but is not one of the discrete valid values the property supports. For example, for a property that coerces values up, if you choose a value of 4 when the adjacent valid values are 1 and 10, the property coerces the value to 10.

Related Topics:

Using Properties and Properties with NI-DMM

Setting Properties Before Reading Properties

Note: Not supported on the PCMCIA-4050 or the PXI/PCI-4060.

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:** .nidmmconfig

initiate

`nidmm.Session.initiate()`

Initiates an acquisition. After you call this method, the DMM leaves the Idle state and enters the Wait-for-Trigger state. If trigger is set to Immediate mode, the DMM begins acquiring measurement data. Use *nidmm.Session.fetch()*, *nidmm.Session.fetch_multi_point()*, or *nidmm.Session.fetch_waveform()* to retrieve the measurement data.

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

lock

`nidmm.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 *nidmm.Session.lock()* method.
- A call to NI-DMM locked the session.
- After a call to the *nidmm.Session.lock()* method returns successfully, no other threads can access the device session until you call the *nidmm.Session.unlock()* method or exit out of the with block when using lock context manager.
- Use the *nidmm.Session.lock()* method and the *nidmm.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 *nidmm.Session.lock()* method within the same thread. To completely unlock the session, you must balance each call to the *nidmm.Session.lock()* method with a call to the *nidmm.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

```python
with nidmm.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, *nidmm.Session.lock()* acts as a context manager and unlock will be called when the *with* block is exited

perform_open_cable_comp

`nidmm.Session.`**`perform_open_cable_comp`**`()`

For the NI 4082 and NI 4072 only, performs the open cable compensation measurements for the current capacitance/inductance range, and returns open cable compensation **Conductance** and **Susceptance** values. You can use the return values of this method as inputs to `nidmm.Session.ConfigureOpenCableCompValues()`.

This method returns an error if the value of the `nidmm.Session.method` property is not set to `CAPACITANCE` (1005) or `INDUCTANCE` (1006).

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

Return type

tuple (conductance, susceptance)

WHERE

conductance (float):

conductance is the measured value of open cable compensation **conductance**.

susceptance (float):

susceptance is the measured value of open cable compensation **susceptance**.

perform_short_cable_comp

`nidmm.Session.`**`perform_short_cable_comp`**`()`

Performs the short cable compensation measurements for the current capacitance/inductance range, and returns short cable compensation **Resistance** and **Reactance** values. You can use the return values of this method as inputs to `nidmm.Session.ConfigureShortCableCompValues()`.

This method returns an error if the value of the `nidmm.Session.method` property is not set to `CAPACITANCE` (1005) or `INDUCTANCE` (1006).

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

Return type

tuple (resistance, reactance)

WHERE

resistance (float):

resistance is the measured value of short cable compensation **resistance**.

reactance (float):

reactance is the measured value of short cable compensation **reactance**.

read

`nidmm.Session.`**`read`**(*maximum_time=hightime.timedelta(milliseconds=-1)*)

Acquires a single measurement and returns the measured value.

Parameters **`maximum_time`** (*hightime.timedelta, datetime.timedelta, or int in milliseconds*) – Specifies the **maximum_time** allowed for this method to complete in milliseconds. If the method does not complete within this time interval, the method returns the NIDMM_ERROR_MAX_TIME_EXCEEDED error code. This may happen if an external trigger has not been received, or if the specified timeout is not long enough for the acquisition to complete.

The valid range is 0–86400000. The default value is `NIDMM_VAL_TIME_LIMIT_AUTO` (-1). The DMM calculates the timeout automatically.

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.

Return type float

Returns The measured value returned from the DMM.

read_multi_point

`nidmm.Session.`**`read_multi_point`**(*array_size*, *maximum_time=hightime.timedelta(milliseconds=-1)*)

Acquires multiple measurements and returns an array of measured values. The number of measurements the DMM makes is determined by the values you specify for the **Trigger_Count** and **Sample_Count** parameters in *`nidmm.Session.configure_multi_point()`*.

Parameters

- **`array_size`** (*int*) – Specifies the number of measurements to acquire. The maximum number of measurements for a finite acquisition is the (**Trigger Count** x **Sample Count**) parameters in *`nidmm.Session.configure_multi_point()`*.

  For continuous acquisitions, up to 100,000 points can be returned at once. The number of measurements can be a subset. The valid range is any positive ViInt32. The default value is 1.

- **`maximum_time`** (*hightime.timedelta, datetime.timedelta, or int in milliseconds*) – Specifies the **maximum_time** allowed for this method to complete in milliseconds. If the method does not complete within this time interval, the method returns the NIDMM_ERROR_MAX_TIME_EXCEEDED error code. This may happen if an external trigger has not been received, or if the specified timeout is not long enough for the acquisition to complete.

  The valid range is 0–86400000. The default value is `NIDMM_VAL_TIME_LIMIT_AUTO` (-1). The DMM calculates the timeout automatically.

  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.

Return type

tuple (reading_array, actual_number_of_points)

WHERE

reading_array (array.array("d")):

An array of measurement values.

Note: The size of the **Reading_Array** must be at least the size that you specify for the **Array_Size** parameter.

actual_number_of_points (int):

Indicates the number of measured values actually retrieved from the DMM.

read_status

`nidmm.Session.`**`read_status`**`()`

Returns measurement backlog and acquisition status. Use this method to determine how many measurements are available before calling *`nidmm.Session.fetch()`*, *`nidmm.Session.fetch_multi_point()`*, or *`nidmm.Session.fetch_waveform()`*.

Note: The NI 4050 is not supported.

Return type

tuple (acquisition_backlog, acquisition_status)

WHERE

acquisition_backlog (int):

The number of measurements available to be read. If the backlog continues to increase, data is eventually overwritten, resulting in an error.

Note: On the NI 4060, the **Backlog** does not increase when autoranging. On the NI 4065, the **Backlog** does not increase when Range is set to AUTO RANGE ON (-1), or before the first point is fetched when Range is set to AUTO RANGE ONCE (-3). These behaviors are due to the autorange model of the devices.

acquisition_status (*`nidmm.AcquisitionStatus`*):

Indicates status of the acquisition. The following table shows the acquisition states:

0	Running
1	Finished with backlog
2	Finished with no backlog
3	Paused
4	No acquisition in progress

read_waveform

`nidmm.Session.`**`read_waveform`**(*array_size*, *maximum_time=hightime.timedelta(milliseconds=-1)*)

For the NI 4080/4081/4082 and the NI 4070/4071/4072, acquires a waveform and returns data as an array of values or as a waveform data type. The number of elements in the **Waveform_Array** is determined by the values you specify for the **Waveform_Points** parameter in *`nidmm.Session.configure_waveform_acquisition()`*.

Parameters

- **`array_size`** (*`int`*) – Specifies the number of waveform points to return. You specify the total number of points that the DMM acquires in the **Waveform Points** parameter of *`nidmm.Session.configure_waveform_acquisition()`*. The default value is 1.
- **`maximum_time`** (*`hightime.timedelta, datetime.timedelta, or int in milliseconds`*) – Specifies the **maximum_time** allowed for this method to complete in milliseconds. If the method does not complete within this time interval, the method returns the NIDMM_ERROR_MAX_TIME_EXCEEDED error code. This may happen if an external trigger has not been received, or if the specified timeout is not long enough for the acquisition to complete.

  The valid range is 0–86400000. The default value is `NIDMM_VAL_TIME_LIMIT_AUTO` (-1). The DMM calculates the timeout automatically.

  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.

Return type

tuple (waveform_array, actual_number_of_points)

WHERE

waveform_array (array.array("d")):

An array of measurement values.

Note: The size of the **Waveform_Array** must be at least the size that you specify for the **Array_Size** parameter.

actual_number_of_points (int):

Indicates the number of measured values actually retrieved from the DMM.

reset

`nidmm.Session.`**`reset`**()

Resets the instrument to a known state and sends initialization commands to the instrument. The initialization commands set instrument settings to the state necessary for the operation of the instrument driver.

reset_with_defaults

`nidmm.Session.reset_with_defaults()`

Resets the instrument to a known state and sends initialization commands to the DMM. The initialization commands set the DMM settings to the state necessary for the operation of NI-DMM. All user-defined default values associated with a logical name are applied after setting the DMM.

self_cal

`nidmm.Session.self_cal()`

For the NI 4080/4081/4082 and the NI 4070/4071/4072, executes the self-calibration routine to maintain measurement accuracy.

Note: This method calls `nidmm.Session.reset()`, and any configurations previous to the call will be lost. All properties will be set to their default values after the call returns.

self_test

`nidmm.Session.self_test()`

Performs a self-test on the DMM to ensure that the DMM is functioning properly. Self-test does not calibrate the DMM. Zero indicates success.

On the NI 4080/4082 and NI 4070/4072, the error code 1013 indicates that you should check the fuse and replace it, if necessary.

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

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

Note: Self-test does not check the fuse on the NI 4065, NI 4071, and NI 4081. Hence, even if the fuse is blown on the device, self-test does not return error code 1013.

Note: This method calls `nidmm.Session.reset()`, and any configurations previous to the call will be lost. All properties will be set to their default values after the call returns.

send_software_trigger

`nidmm.Session.send_software_trigger()`

Sends a command to trigger the DMM. Call this method if you have configured either the `nidmm.Session.trigger_source` or `nidmm.Session.sample_trigger` properties. If the `nidmm.Session.trigger_source` and/or `nidmm.Session.sample_trigger` properties are set to `NIDMM_VAL_EXTERNAL` or `NIDMM_VAL_TTL`*n*, you can use this method to override the trigger source that you configured and trigger the device. The NI 4050 and NI 4060 are not supported.

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.

unlock

`nidmm.Session.unlock()`

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

Properties

ac_max_freq

`nidmm.Session.ac_max_freq`

Specifies the maximum frequency component of the input signal for AC measurements. This property is used only for error checking and verifies that the value of this parameter is less than the maximum frequency of the device. This property affects the DMM only when you set the `nidmm.Session.method` property to AC measurements. The valid range is 1 Hz-300 kHz for the NI 4070/4071/4072, 10 Hz-100 kHz for the NI 4065, and 20 Hz-25 kHz for the NI 4050 and NI 4060.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Max Frequency**
- C Attribute: **NIDMM_ATTR_AC_MAX_FREQ**

ac_min_freq

`nidmm.Session.ac_min_freq`

Specifies the minimum frequency component of the input signal for AC measurements. This property affects the DMM only when you set the `nidmm.Session.method` property to AC measurements. The valid range is 1 Hz-300 kHz for the NI 4070/4071/4072, 10 Hz-100 kHz for the NI 4065, and 20 Hz-25 kHz for the NI 4050 and NI 4060.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Min Frequency**
- C Attribute: **NIDMM_ATTR_AC_MIN_FREQ**

adc_calibration

`nidmm.Session.`**`adc_calibration`**

For the NI 4070/4071/4072 only, specifies the ADC calibration mode.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ADCCalibration
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:ADC Calibration**
- C Attribute: **NIDMM_ATTR_ADC_CALIBRATION**

aperture_time

`nidmm.Session.`**`aperture_time`**

Specifies the measurement aperture time for the current configuration. Aperture time is specified in units set by *`nidmm.Session.aperture_time_units`*. To override the default aperture, set this property to the desired aperture time after calling `nidmm.Session.ConfigureMeasurement()`. To return to the default, set this property to `NIDMM_VAL_APERTURE_TIME_AUTO` (-1). On the NI 4070/4071/4072, the minimum aperture time is 8.89 usec, and the maximum aperture time is 149 sec. Any number of powerline cycles (PLCs) within the minimum and maximum ranges is allowed on the NI 4070/4071/4072. On the NI 4065 the minimum aperture time is 333 µs, and the maximum aperture time is 78.2 s. If setting the number of averages directly, the total measurement time is aperture time X the number of averages, which must be less than 72.8 s. The aperture times allowed are 333 µs, 667 µs, or multiples of 1.11 ms-for example 1.11 ms, 2.22 ms, 3.33 ms, and so on. If you set an aperture time other than 333 µs, 667 µs, or multiples of 1.11 ms, the value will be coerced up to the next supported aperture time. On the NI 4060, when the powerline frequency is 60 Hz, the PLCs allowed are 1 PLC, 6 PLC, 12 PLC, and 120 PLC. When the powerline frequency is 50 Hz, the PLCs allowed are 1 PLC, 5 PLC, 10 PLC, and 100 PLC.

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.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Advanced:Aperture Time**
- C Attribute: **NIDMM_ATTR_APERTURE_TIME**

aperture_time_units

`nidmm.Session.`**`aperture_time_units`**

Specifies the units of aperture time for the current configuration. The NI 4060 does not support an aperture time set in seconds.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ApertureTimeUnits
Permissions	read-write

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

- LabVIEW Property: **Configuration:Advanced:Aperture Time Units**
- C Attribute: **NIDMM_ATTR_APERTURE_TIME_UNITS**

auto_range_value

`nidmm.Session.`**`auto_range_value`**

Specifies the value of the range. If auto ranging, shows the actual value of the active range. The value of this property is set during a read operation.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only

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

- LabVIEW Property: **Configuration:Auto Range Value**
- C Attribute: **NIDMM_ATTR_AUTO_RANGE_VALUE**

auto_zero

`nidmm.Session.auto_zero`

Specifies the AutoZero mode. The NI 4050 is not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.AutoZero
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Auto Zero**
- C Attribute: **NIDMM_ATTR_AUTO_ZERO**

buffer_size

`nidmm.Session.buffer_size`

Size in samples of the internal data buffer. Maximum is 134,217,727 (OX7FFFFFF) samples. When set to `NIDMM_VAL_BUFFER_SIZE_AUTO` (-1), NI-DMM chooses the buffer size.

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	int
Permissions	read-write

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

- LabVIEW Property: **Multi Point Acquisition:Advanced:Buffer Size**
- C Attribute: **NIDMM_ATTR_BUFFER_SIZE**

cable_comp_type

`nidmm.Session.cable_comp_type`

For the NI 4072 only, the type of cable compensation that is applied to the current capacitance or inductance measurement for the current range. Changing the method or the range through this property or through *`nidmm.Session.configure_measurement_digits()`* resets the value of this property to the default value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.CableCompensationType
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Capacitance and Inductance:Cable Compensation Type**
- C Attribute: **NIDMM_ATTR_CABLE_COMP_TYPE**

channel_count

`nidmm.Session.`**`channel_count`**

Indicates the number of channels that the specific instrument driver supports. For each property for which the IVI_VAL_MULTI_CHANNEL flag property is set, 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

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

- LabVIEW Property: **Inherent IVI Attributes:Instrument Capabilities:Channel Count**
- C Attribute: **NIDMM_ATTR_CHANNEL_COUNT**

current_source

`nidmm.Session.`**`current_source`**

Specifies the current source provided during diode measurements. The NI 4050 and NI 4060 are not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Current Source**
- C Attribute: **NIDMM_ATTR_CURRENT_SOURCE**

dc_bias

`nidmm.Session.dc_bias`

For the NI 4072 only, controls the available DC bias for capacitance measurements.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Capacitance and Inductance:Advanced:DC Bias**
- C Attribute: **NIDMM_ATTR_DC_BIAS**

dc_noise_rejection

`nidmm.Session.dc_noise_rejection`

Specifies the DC noise rejection mode. The NI 4050 and NI 4060 are not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.DCNoiseRejection
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:DC Noise Rejection**
- C Attribute: **NIDMM_ATTR_DC_NOISE_REJECTION**

driver_setup

`nidmm.Session.driver_setup`

This property indicates the Driver Setup string that the user specified when initializing the driver. Some cases exist where the end-user must specify instrument driver options at initialization time. An example of this is specifying a particular instrument model from among a family of instruments that the driver supports. This is useful when using simulation. The end-user can specify driver-specific options through the DriverSetup keyword in the optionsString parameter to the niDMM Init With Options.vi. If the user does not specify a Driver Setup string, this property returns an empty string.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:User Options:Driver Setup**
- C Attribute: **NIDMM_ATTR_DRIVER_SETUP**

freq_voltage_auto_range

nidmm.Session.**freq_voltage_auto_range**

For the NI 4070/4071/4072 only, specifies the value of the frequency voltage range. If Auto Ranging, shows the actual value of the active frequency voltage range. If not Auto Ranging, the value of this property is the same as that of *nidmm.Session.freq_voltage_range*.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only

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

- LabVIEW Property: **Configuration:Measurement Options:Frequency Voltage Auto Range Value**
- C Attribute: **NIDMM_ATTR_FREQ_VOLTAGE_AUTO_RANGE**

freq_voltage_range

nidmm.Session.**freq_voltage_range**

Specifies the maximum amplitude of the input signal for frequency measurements.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Frequency Voltage Range**
- C Attribute: **NIDMM_ATTR_FREQ_VOLTAGE_RANGE**

function

nidmm.Session.**function**

Specifies the measurement method. Refer to the nidmm.Session.method topic in the NI Digital Multimeters Help for device-specific information. If you are setting this property directly, you must also set the *nidmm.Session.operation_mode* property, which controls whether the DMM takes standard single or multipoint measurements, or acquires a waveform. If you are programming properties directly, you must set the *nidmm.Session.operation_mode* property before setting other configuration properties. If the *nidmm.Session.operation_mode* property is set to *WAVEFORM*, the only valid method types are `WAVEFORM_VOLTAGE` and `WAVEFORM_CURRENT`. Set the *nidmm.Session.operation_mode* property to *IVIDMM* to set all other method values.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.Function
Permissions	read-write

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

- LabVIEW Property: **Configuration:Function**
- C Attribute: **NIDMM_ATTR_FUNCTION**

input_resistance

`nidmm.Session.`**`input_resistance`**

Specifies the input resistance of the instrument. The NI 4050 and NI 4060 are not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Input Resistance**
- C Attribute: **NIDMM_ATTR_INPUT_RESISTANCE**

instrument_firmware_revision

`nidmm.Session.`**`instrument_firmware_revision`**

A string containing the instrument firmware revision number.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Instrument Identification:Instrument Firmware Revision**
- C Attribute: **NIDMM_ATTR_INSTRUMENT_FIRMWARE_REVISION**

instrument_manufacturer

`nidmm.Session.`**`instrument_manufacturer`**

A string containing the manufacturer of the instrument.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Instrument Identification:Instrument Manufacturer**
- C Attribute: **NIDMM_ATTR_INSTRUMENT_MANUFACTURER**

instrument_model

`nidmm.Session.`**`instrument_model`**

A string containing the instrument model.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Instrument Identification:Instrument Model**
- C Attribute: **NIDMM_ATTR_INSTRUMENT_MODEL**

instrument_product_id

`nidmm.Session.`**`instrument_product_id`**

The PCI product ID.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Instrument Identification:Instrument Product ID**
- C Attribute: **NIDMM_ATTR_INSTRUMENT_PRODUCT_ID**

io_resource_descriptor

nidmm.Session.**io_resource_descriptor**

A string containing the resource descriptor of the instrument.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Advanced Session Information:I/O Resource Descriptor**
- C Attribute: **NIDMM_ATTR_IO_RESOURCE_DESCRIPTOR**

lc_calculation_model

nidmm.Session.**lc_calculation_model**

For the NI 4072 only, specifies the type of algorithm that the measurement processing uses for capacitance and inductance measurements.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.LCCalculationModel
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Capacitance and Inductance:Advanced:Calculation Model**
- C Attribute: **NIDMM_ATTR_LC_CALCULATION_MODEL**

lc_number_meas_to_average

nidmm.Session.**lc_number_meas_to_average**

For the NI 4072 only, specifies the number of LC measurements that are averaged to produce one reading.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Capacitance and Inductance:Number of LC Measurements To Average**
- C Attribute: **NIDMM_ATTR_LC_NUMBER_MEAS_TO_AVERAGE**

logical_name

nidmm.Session.**logical_name**

A string containing the logical name of the instrument.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Advanced Session Information:Logical Name**
- C Attribute: **NIDMM_ATTR_LOGICAL_NAME**

meas_complete_dest

nidmm.Session.**meas_complete_dest**

Specifies the destination of the measurement complete (MC) signal. The NI 4050 is not supported. To determine which values are supported by each device, refer to the LabWindows/CVI Trigger Routing section in the NI Digital Multimeters Help.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.MeasurementCompleteDest
Permissions	read-write

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

- LabVIEW Property: **Trigger:Measurement Complete Dest**
- C Attribute: **NIDMM_ATTR_MEAS_COMPLETE_DEST**

number_of_averages

nidmm.Session.**number_of_averages**

Specifies the number of averages to perform in a measurement. For the NI 4070/4071/4072, applies only when the aperture time is not set to AUTO and Auto Zero is ON. The default is 1. The NI 4050 and NI 4060 are not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write

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

- LabVIEW Property: **Configuration:Advanced:Number Of Averages**
- C Attribute: **NIDMM_ATTR_NUMBER_OF_AVERAGES**

offset_comp_ohms

nidmm.Session.**offset_comp_ohms**

For the NI 4070/4071/4072 only, enables or disables offset compensated ohms.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Offset Compensated Ohms**
- C Attribute: **NIDMM_ATTR_OFFSET_COMP_OHMS**

open_cable_comp_conductance

nidmm.Session.**open_cable_comp_conductance**

For the NI 4072 only, specifies the active part (conductance) of the open cable compensation. The valid range is any real number greater than 0. The default value (-1.0) indicates that compensation has not taken place. Changing the method or the range through this property or through `nidmm.Session.configure_measurement_digits()` resets the value of this property to the default value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Capacitance and Inductance:Open Cable Compensation Values:Conductance**
- C Attribute: **NIDMM_ATTR_OPEN_CABLE_COMP_CONDUCTANCE**

open_cable_comp_susceptance

`nidmm.Session.`**`open_cable_comp_susceptance`**

For the NI 4072 only, specifies the reactive part (susceptance) of the open cable compensation. The valid range is any real number greater than 0. The default value (-1.0) indicates that compensation has not taken place. Changing the method or the range through this property or through `nidmm.Session.configure_measurement_digits()` resets the value of this property to the default value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Capacitance and Inductance:Open Cable Compensation Values:Susceptance**
- C Attribute: **NIDMM_ATTR_OPEN_CABLE_COMP_SUSCEPTANCE**

operation_mode

`nidmm.Session.`**`operation_mode`**

Specifies how the NI 4065 and NI 4070/4071/4072 acquire data. When you call `nidmm.Session.configure_measurement_digits()`, NI-DMM sets this property to `IVIDMM`. When you call `nidmm.Session.configure_waveform_acquisition()`, NI-DMM sets this property to `WAVEFORM`. If you are programming properties directly, you must set this property before setting other configuration properties.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.OperationMode
Permissions	read-write

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

- LabVIEW Property: **Configuration:Advanced:Operation Mode**
- C Attribute: **NIDMM_ATTR_OPERATION_MODE**

powerline_freq

nidmm.Session.**powerline_freq**

Specifies the powerline frequency. The NI 4050 and NI 4060 use this value to select an aperture time to reject powerline noise by selecting the appropriate internal sample clock and filter. The NI 4065 and NI 4070/4071/4072 use this value to select a timebase for setting the *nidmm.Session.aperture_time* property in powerline cycles (PLCs). After configuring powerline frequency, set the *nidmm.Session.aperture_time_units* property to PLCs. When setting the *nidmm.Session.aperture_time* property, select the number of PLCs for the powerline frequency. For example, if powerline frequency = 50 Hz (or 20ms) and aperture time in PLCs = 5, then aperture time in Seconds = 20ms * 5 PLCs = 100 ms. Similarly, if powerline frequency = 60 Hz (or 16.667 ms) and aperture time in PLCs = 6, then aperture time in Seconds = 16.667 ms * 6 PLCs = 100 ms.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Powerline Frequency**
- C Attribute: **NIDMM_ATTR_POWERLINE_FREQ**

range

nidmm.Session.**range**

Specifies the measurement range. Use positive values to represent the absolute value of the maximum expected measurement. The value is in units appropriate for the current value of the `nidmm.Session.method` property. For example, if `nidmm.Session.method` is set to `NIDMM_VAL_VOLTS`, the units are volts. The NI 4050 and NI 4060 only support Auto Range when the trigger and sample trigger is set to IMMEDIATE. `NIDMM_VAL_AUTO_RANGE_ON` -1.0 NI-DMM performs an Auto Range before acquiring the measurement. `NIDMM_VAL_AUTO_RANGE_OFF` -2.0 NI-DMM sets the Range to the current *nidmm.Session.auto_range_value* and uses this range for all subsequent measurements until the measurement configuration is changed. `NIDMM_VAL_AUTO_RANGE_ONCE` -3.0 NI-DMM performs an Auto Range before acquiring the next measurement. The *nidmm.Session.*

auto_range_value is stored and used for all subsequent measurements until the measurement configuration is changed.

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	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Range**
- C Attribute: **NIDMM_ATTR_RANGE**

resolution_absolute

nidmm.Session.**resolution_absolute**

Specifies the measurement resolution in absolute units. Setting this property to higher values increases the measurement accuracy. Setting this property to lower values increases the measurement speed. NI-DMM ignores this property for capacitance and inductance measurements on the NI 4072. To achieve better resolution for such measurements, use the *nidmm.Session.lc_number_meas_to_average* property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Absolute Resolution**
- C Attribute: **NIDMM_ATTR_RESOLUTION_ABSOLUTE**

resolution_digits

nidmm.Session.**resolution_digits**

Specifies the measurement resolution in digits. Setting this property to higher values increases the measurement accuracy. Setting this property to lower values increases the measurement speed. NI-DMM ignores this property for capacitance and inductance measurements on the NI 4072. To achieve better resolution for such measurements, use the *nidmm.Session.lc_number_meas_to_average* property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Digits Resolution**
- C Attribute: **NIDMM_ATTR_RESOLUTION_DIGITS**

sample_count

`nidmm.Session.`**`sample_count`**

Specifies the number of measurements the DMM takes each time it receives a trigger in a multiple point acquisition.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write

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

- LabVIEW Property: **Multi Point Acquisition:Sample Count**
- C Attribute: **NIDMM_ATTR_SAMPLE_COUNT**

sample_interval

`nidmm.Session.`**`sample_interval`**

Specifies the amount of time in seconds the DMM waits between measurement cycles. This property only applies when the *`nidmm.Session.sample_trigger`* property is set to INTERVAL. On the NI 4060, the value for this property is used as the settling time. When this property is set to 0, the NI 4060 does not settle between measurement cycles. The onboard timing resolution is 1 µs on the NI 4060. The NI 4065 and NI 4070/4071/4072 use the value specified in this property as additional delay. On the NI 4065 and NI 4070/4071/4072, the onboard timing resolution is 34.72 ns and the valid range is 0-149 s. Only positive values are valid when setting the sample interval. The NI 4050 is not supported.

The following table lists the characteristics of this property.

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

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

- LabVIEW Property: **Multi Point Acquisition:Sample Interval**
- C Attribute: **NIDMM_ATTR_SAMPLE_INTERVAL**

sample_trigger

nidmm.Session.**sample_trigger**

Specifies the sample trigger source. To determine which values are supported by each device, refer to the LabWindows/CVI Trigger Routing section in the NI Digital Multimeters Help.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.SampleTrigger
Permissions	read-write

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

- LabVIEW Property: **Multi Point Acquisition:Sample Trigger**
- C Attribute: **NIDMM_ATTR_SAMPLE_TRIGGER**

serial_number

nidmm.Session.**serial_number**

A string containing the serial number of the instrument. This property corresponds to the serial number label that is attached to most products.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Instrument Identification:Instrument Serial Number**
- C Attribute: **NIDMM_ATTR_SERIAL_NUMBER**

settle_time

nidmm.Session.**settle_time**

Specifies the settling time in seconds. To override the default settling time, set this property. To return to the default, set this property to `NIDMM_VAL_SETTLE_TIME_AUTO` (-1). The NI 4050 and NI 4060 are not supported.

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	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write

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

- LabVIEW Property: **Configuration:Advanced:Settle Time**
- C Attribute: **NIDMM_ATTR_SETTLE_TIME**

short_cable_comp_reactance

`nidmm.Session.`**`short_cable_comp_reactance`**

For the NI 4072 only, represents the reactive part (reactance) of the short cable compensation. The valid range is any real number greater than 0. The default value (-1) indicates that compensation has not taken place. Changing the method or the range through this property or through *`nidmm.Session.configure_measurement_digits()`* resets the value of this property to the default value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Capacitance and Inductance:Short Cable Compensation Values:Reactance**
- C Attribute: **NIDMM_ATTR_SHORT_CABLE_COMP_REACTANCE**

short_cable_comp_resistance

`nidmm.Session.`**`short_cable_comp_resistance`**

For the NI 4072 only, represents the active part (resistance) of the short cable compensation. The valid range is any real number greater than 0. The default value (-1) indicates that compensation has not taken place. Changing the method or the range through this property or through *`nidmm.Session.configure_measurement_digits()`* resets the value of this property to the default value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Capacitance and Inductance:Short Cable Compensation Values:Resistance**
- C Attribute: **NIDMM_ATTR_SHORT_CABLE_COMP_RESISTANCE**

simulate

`nidmm.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 Get and Set 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 (0). Use the `nidmm.Session.__init__()` method to override this setting. Simulate can only be set within the InitWithOptions method. The property value cannot be changed outside of the method.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write

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

- LabVIEW Property: **Inherent IVI Attributes:User Options:Simulate**
- C Attribute: **NIDMM_ATTR_SIMULATE**

specific_driver_description

`nidmm.Session.`**`specific_driver_description`**

A string containing a description of the specific driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Specific Driver Identification:Specific Driver Description**

- C Attribute: **NIDMM_ATTR_SPECIFIC_DRIVER_DESCRIPTION**

specific_driver_major_version

`nidmm.Session.`**`specific_driver_major_version`**

Returns the major version number of this instrument driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Version Info:Specific Driver Major Version**
- C Attribute: **NIDMM_ATTR_SPECIFIC_DRIVER_MAJOR_VERSION**

specific_driver_minor_version

`nidmm.Session.`**`specific_driver_minor_version`**

The minor version number of this instrument driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Version Info:Specific Driver Minor Version**
- C Attribute: **NIDMM_ATTR_SPECIFIC_DRIVER_MINOR_VERSION**

specific_driver_revision

`nidmm.Session.`**`specific_driver_revision`**

A string that contains additional version information about this specific instrument driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Version Info:Specific Driver Revision**
- C Attribute: **NIDMM_ATTR_SPECIFIC_DRIVER_REVISION**

specific_driver_vendor

`nidmm.Session.`**`specific_driver_vendor`**

A string containing the vendor of the specific driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Specific Driver Identification:Specific Driver Vendor**
- C Attribute: **NIDMM_ATTR_SPECIFIC_DRIVER_VENDOR**

supported_instrument_models

`nidmm.Session.`**`supported_instrument_models`**

A string containing the instrument models supported by the specific driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

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

- LabVIEW Property: **Inherent IVI Attributes:Specific Driver Capabilities:Supported Instrument Models**
- C Attribute: **NIDMM_ATTR_SUPPORTED_INSTRUMENT_MODELS**

temp_rtd_a

`nidmm.Session.`**`temp_rtd_a`**

Specifies the Callendar-Van Dusen A coefficient for RTD scaling when the RTD Type property is set to Custom. The default value is 3.9083e-3 (Pt3851).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Resistance Temperature Detector:RTD A**
- C Attribute: **NIDMM_ATTR_TEMP_RTD_A**

temp_rtd_b

`nidmm.Session.`**`temp_rtd_b`**

Specifies the Callendar-Van Dusen B coefficient for RTD scaling when the RTD Type property is set to Custom. The default value is -5.775e-7(Pt3851).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Resistance Temperature Detector:RTD B**
- C Attribute: **NIDMM_ATTR_TEMP_RTD_B**

temp_rtd_c

`nidmm.Session.`**`temp_rtd_c`**

Specifies the Callendar-Van Dusen C coefficient for RTD scaling when the RTD Type property is set to Custom. The default value is -4.183e-12(Pt3851).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Resistance Temperature Detector:RTD C**
- C Attribute: **NIDMM_ATTR_TEMP_RTD_C**

temp_rtd_res

`nidmm.Session.temp_rtd_res`

Specifies the RTD resistance at 0 degrees Celsius. This applies to all supported RTDs, including custom RTDs. The default value is 100 (?).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Resistance Temperature Detector:RTD Resistance**
- C Attribute: **NIDMM_ATTR_TEMP_RTD_RES**

temp_rtd_type

`nidmm.Session.temp_rtd_type`

Specifies the type of RTD used to measure temperature. The default value is *PT3851*. Refer to the *nidmm.Session.temp_rtd_type* topic in the NI Digital Multimeters Help for additional information about defined values.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.RTDType
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Resistance Temperature Detector:RTD Type**
- C Attribute: **NIDMM_ATTR_TEMP_RTD_TYPE**

temp_tc_fixed_ref_junc

`nidmm.Session.temp_tc_fixed_ref_junc`

Specifies the reference junction temperature when a fixed reference junction is used to take a thermocouple measurement. The default value is 25.0 (°C).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Thermocouple:Fixed Reference Junction**
- C Attribute: **NIDMM_ATTR_TEMP_TC_FIXED_REF_JUNC**

temp_tc_ref_junc_type

`nidmm.Session.`**`temp_tc_ref_junc_type`**

Specifies the type of reference junction to be used in the reference junction compensation of a thermocouple. The only supported value, `NIDMM_VAL_TEMP_REF_JUNC_FIXED`, is fixed.

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	enums.ThermocoupleReferenceJunctionType
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Thermocouple:Reference Junction Type**
- C Attribute: **NIDMM_ATTR_TEMP_TC_REF_JUNC_TYPE**

temp_tc_type

`nidmm.Session.`**`temp_tc_type`**

Specifies the type of thermocouple used to measure the temperature. The default value is `J`.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ThermocoupleType
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Thermocouple:Thermocouple Type**
- C Attribute: **NIDMM_ATTR_TEMP_TC_TYPE**

temp_thermistor_a

`nidmm.Session.`**`temp_thermistor_a`**

Specifies the Steinhart-Hart A coefficient for thermistor scaling when the Thermistor Type property is set to Custom. The default value is 0.0010295 (44006).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Thermistor:Thermistor A**
- C Attribute: **NIDMM_ATTR_TEMP_THERMISTOR_A**

temp_thermistor_b

`nidmm.Session.`**`temp_thermistor_b`**

Specifies the Steinhart-Hart B coefficient for thermistor scaling when the Thermistor Type proerty is set to Custom. The default value is 0.0002391 (44006).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

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

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Thermistor:Thermistor B**
- C Attribute: **NIDMM_ATTR_TEMP_THERMISTOR_B**

temp_thermistor_c

`nidmm.Session.`**`temp_thermistor_c`**

Specifies the Steinhart-Hart C coefficient for thermistor scaling when the Thermistor Type property is set to Custom. The default value is 1.568e-7 (44006).

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Thermistor:Thermistor C**
- C Attribute: **NIDMM_ATTR_TEMP_THERMISTOR_C**

---

temp_thermistor_type

`nidmm.Session.`**`temp_thermistor_type`**

Specifies the type of thermistor used to measure the temperature. The default value is *`THERMISTOR_44006`*. Refer to the *`nidmm.Session.temp_thermistor_type`* topic in the NI Digital Multimeters Help for additional information about defined values.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ThermistorType
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Thermistor:Thermistor Type**
- C Attribute: **NIDMM_ATTR_TEMP_THERMISTOR_TYPE**

---

temp_transducer_type

`nidmm.Session.`**`temp_transducer_type`**

Specifies the type of device used to measure the temperature. The default value is `NIDMM_VAL_4_THERMOCOUPLE`.

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	enums.TransducerType
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Configuration:Measurement Options:Temperature:Transducer Type**
- C Attribute: **NIDMM_ATTR_TEMP_TRANSDUCER_TYPE**

---

trigger_count

`nidmm.Session.trigger_count`

Specifies the number of triggers the DMM receives before returning to the Idle state. This property can be set to any positive ViInt32 value for the NI 4065 and NI 4070/4071/4072. The NI 4050 and NI 4060 support this property being set to 1. Refer to the Multiple Point Acquisitions section of the NI Digital Multimeters Help for more information.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Multi Point Acquisition:Trigger Count**
- C Attribute: **NIDMM_ATTR_TRIGGER_COUNT**

trigger_delay

`nidmm.Session.trigger_delay`

Specifies the time (in seconds) that the DMM waits after it has received a trigger before taking a measurement. The default value is AUTO DELAY (-1), which means that the DMM waits an appropriate settling time before taking the measurement. (-1) signifies that AUTO DELAY is on, and (-2) signifies that AUTO DELAY is off. The NI 4065 and NI 4070/4071/4072 use the value specified in this property as additional settling time. For the The NI 4065 and NI 4070/4071/4072, the valid range for Trigger Delay is AUTO DELAY (-1) or 0.0-149.0 seconds and the onboard timing resolution is 34.72 ns. On the NI 4060, if this property is set to 0, the DMM does not settle before taking the measurement. On the NI 4060, the valid range for AUTO DELAY (-1) is 0.0-12.0 seconds and the onboard timing resolution is 100 ms. When using the NI 4050, this property must be set to AUTO DELAY (-1). Use positive values to set the trigger delay in seconds. Valid Range: `NIDMM_VAL_AUTO_DELAY` (-1.0), 0.0-12.0 seconds (NI 4060 only) Default Value: `NIDMM_VAL_AUTO_DELAY`

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	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Trigger:Trigger Delay**
- C Attribute: **NIDMM_ATTR_TRIGGER_DELAY**

trigger_source

`nidmm.Session.`**`trigger_source`**

Specifies the trigger source. When `nidmm.Session._initiate()` is called, the DMM waits for the trigger specified with this property. After it receives the trigger, the DMM waits the length of time specified with the *`nidmm.Session.trigger_delay`* property. The DMM then takes a measurement. This property is not supported on the NI 4050. To determine which values are supported by each device, refer to the LabWindows/CVI Trigger Routing section in the NI Digital Multimeters Help.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TriggerSource
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Trigger:Trigger Source**
- C Attribute: **NIDMM_ATTR_TRIGGER_SOURCE**

waveform_coupling

`nidmm.Session.`**`waveform_coupling`**

For the NI 4070/4071/4072 only, specifies the coupling during a waveform acquisition.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.WaveformCoupling
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Acquisition:Waveform Coupling**
- C Attribute: **NIDMM_ATTR_WAVEFORM_COUPLING**

waveform_points

`nidmm.Session.`**`waveform_points`**

For the NI 4070/4071/4072 only, specifies the number of points to acquire in a waveform acquisition.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Acquisition:Waveform Points**
- C Attribute: **NIDMM_ATTR_WAVEFORM_POINTS**

waveform_rate

`nidmm.Session.`**`waveform_rate`**

For the NI 4070/4071/4072 only, specifies the rate of the waveform acquisition in Samples per second (S/s). The valid Range is 10.0-1,800,000 S/s. Values are coerced to the closest integer divisor of 1,800,000. The default value is 1,800,000.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Waveform Acquisition:Waveform Rate**
- C Attribute: **NIDMM_ATTR_WAVEFORM_RATE**

Session

- *auto_range_value*
- *auto_zero*
- *buffer_size*
- *cable_comp_type*
- *channel_count*
- *current_source*
- *dc_bias*
- *dc_noise_rejection*
- *driver_setup*
- *freq_voltage_auto_range*
- *freq_voltage_range*
- *function*
- *input_resistance*
- *instrument_firmware_revision*
- *instrument_manufacturer*
- *instrument_model*
- *instrument_product_id*
- *io_resource_descriptor*
- *lc_calculation_model*
- *lc_number_meas_to_average*
- *logical_name*
- *meas_complete_dest*
- *number_of_averages*
- *offset_comp_ohms*
- *open_cable_comp_conductance*
- *open_cable_comp_susceptance*
- *operation_mode*
- *powerline_freq*
- *range*
- *resolution_absolute*
- *resolution_digits*
- *sample_count*
- *sample_interval*
- *sample_trigger*
- *serial_number*

Enums

Enums used in NI-DMM

ADCCalibration

class nidmm.**ADCCalibration**

AUTO
The DMM enables or disables ADC calibration for you.

OFF
The DMM does not compensate for changes to the gain.

ON
The DMM measures an internal reference to calculate the correct gain for the measurement.

AcquisitionStatus

class nidmm.**AcquisitionStatus**

RUNNING
Running

FINISHED_WITH_BACKLOG
Finished with **Backlog**

FINISHED_WITH_NO_BACKLOG
Finished with no **Backlog**

PAUSED
Paused

NO_ACQUISITION_IN_PROGRESS
No acquisition in progress

ApertureTimeUnits

class nidmm.**ApertureTimeUnits**

SECONDS
Seconds

POWER_LINE_CYCLES
Powerline Cycles

AutoZero

class nidmm.**AutoZero**

AUTO
The drivers chooses the AutoZero setting based on the configured method and resolution.

OFF
Disables AutoZero.

ON

The DMM internally disconnects the input signal following each measurement and takes a zero reading. It then subtracts the zero reading from the preceding reading.

ONCE

The DMM internally disconnects the input signal for the first measurement and takes a zero reading. It then subtracts the zero reading from the first reading and the following readings.

CableCompensationType

class nidmm.**CableCompensationType**

NONE

No Cable Compensation

OPEN

Open Cable Compensation

SHORT

Short Cable Compensation

OPEN_AND_SHORT

Open and Short Cable Compensation

DCNoiseRejection

class nidmm.**DCNoiseRejection**

AUTO

The driver chooses the DC noise rejection setting based on the configured method and resolution.

NORMAL

NI-DMM weighs all samples equally.

SECOND_ORDER

NI-DMM weighs the samples taken in the middle of the aperture time more than samples taken at the beginning and the end of the measurement using a triangular weighing method.

HIGH_ORDER

NI-DMM weighs the samples taken in the middle of the aperture time more than samples taken at the beginning and the end of the measurement using a bell-curve weighing method.

Function

class nidmm.**Function**

DC_VOLTS

DC Voltage

AC_VOLTS

AC Voltage

DC_CURRENT

DC Current

AC_CURRENT
AC Current

TWO_WIRE_RES
2-Wire Resistance

FOUR_WIRE_RES
4-Wire Resistance

FREQ
Frequency

PERIOD
Period

TEMPERATURE
NI 4065, NI 4070/4071/4072, and NI 4080/4081/4182 supported.

AC_VOLTS_DC_COUPLED
AC Voltage with DC Coupling

DIODE
Diode

WAVEFORM_VOLTAGE
Waveform voltage

WAVEFORM_CURRENT
Waveform current

CAPACITANCE
Capacitance

INDUCTANCE
Inductance

LCCalculationModel

class nidmm.**LCCalculationModel**

AUTO
NI-DMM chooses the algorithm based on method and range

SERIES
NI-DMM uses the series impedance model to calculate capacitance and inductance

PARALLEL
NI-DMM uses the parallel admittance model to calculate capacitance and inductance

MeasurementCompleteDest

class nidmm.**MeasurementCompleteDest**

NONE
No Trigger

EXTERNAL
AUX I/O Connector

PXI_TRIG0
PXI Trigger Line 0

PXI_TRIG1
PXI Trigger Line 1

PXI_TRIG2
PXI Trigger Line 2

PXI_TRIG3
PXI Trigger Line 3

PXI_TRIG4
PXI Trigger Line 4

PXI_TRIG5
PXI Trigger Line 5

PXI_TRIG6
PXI Trigger Line 6

PXI_TRIG7
PXI Trigger Line 7

LBR_TRIG0
Internal Trigger Line of a PXI/SCXI Combination Chassis

OperationMode

class nidmm.**OperationMode**

IVIDMM
IviDmm Mode

WAVEFORM
Waveform acquisition mode

RTDType

class nidmm.**RTDType**

CUSTOM
Performs Callendar-Van Dusen RTD scaling with the user-specified A, B, and C coefficients.

PT3750
Performs scaling for a Pt 3750 RTD.

PT3851
Performs scaling for a Pt 3851 RTD.

PT3911
Performs scaling for a Pt 3911 RTD.

PT3916
Performs scaling for a Pt 3916 RTD.

PT3920
Performs scaling for a Pt 3920 RTD.

PT3928
Performs scaling for a Pt 3928 RTD.

SampleTrigger

class nidmm.**SampleTrigger**

IMMEDIATE
No Trigger

EXTERNAL
AUX I/O Connector Trigger Line 0

SOFTWARE_TRIG
Software Trigger

INTERVAL
Interval Trigger

PXI_TRIG0
PXI Trigger Line 0

PXI_TRIG1
PXI Trigger Line 1

PXI_TRIG2
PXI Trigger Line 2

PXI_TRIG3
PXI Trigger Line 3

PXI_TRIG4
PXI Trigger Line 4

PXI_TRIG5
PXI Trigger Line 5

PXI_TRIG6
PXI Trigger Line 6

PXI_TRIG7
PXI Trigger Line 7

PXI_STAR
PXI Star Trigger Line

AUX_TRIG1
AUX I/0 Connector Trigger Line 1

LBR_TRIG1
Internal Trigger Line of a PXI/SCXI Combination Chassis

ThermistorType

class nidmm.**ThermistorType**

CUSTOM
Custom

THERMISTOR_44004
44004

THERMISTOR_44006
44006

THERMISTOR_44007
44007

ThermocoupleReferenceJunctionType

`class nidmm.ThermocoupleReferenceJunctionType`

FIXED
Thermocouple reference juction is fixed at the user-specified temperature.

ThermocoupleType

`class nidmm.ThermocoupleType`

B
Thermocouple type B

E
Thermocouple type E

J
Thermocouple type J

K
Thermocouple type K

N
Thermocouple type N

R
Thermocouple type R

S
Thermocouple type S

T
Thermocouple type T

TransducerType

`class nidmm.TransducerType`

THERMOCOUPLE
Thermocouple

THERMISTOR
Thermistor

TWO_WIRE_RTD
2-wire RTD

FOUR_WIRE_RTD
4-wire RTD

TriggerSource

class nidmm.**TriggerSource**

IMMEDIATE
No Trigger

EXTERNAL
AUX I/O Connector Trigger Line 0

SOFTWARE_TRIG
Software Trigger

PXI_TRIG0
PXI Trigger Line 0

PXI_TRIG1
PXI Trigger Line 1

PXI_TRIG2
PXI Trigger Line 2

PXI_TRIG3
PXI Trigger Line 3

PXI_TRIG4
PXI Trigger Line 4

PXI_TRIG5
PXI Trigger Line 5

PXI_TRIG6
PXI Trigger Line 6

PXI_TRIG7
PXI Trigger Line 7

PXI_STAR
PXI Star Trigger Line

AUX_TRIG1
AUX I/O Connector Trigger Line 1

LBR_TRIG1
Internal Trigger Line of a PXI/SCXI Combination Chassis

WaveformCoupling

class nidmm.**WaveformCoupling**

AC
AC Coupled

DC
DC Coupled

Exceptions and Warnings

Error

exception `nidmm.errors.`**Error**
Base exception type that all NI-DMM exceptions derive from

DriverError

exception `nidmm.errors.`**DriverError**
An error originating from the NI-DMM driver

UnsupportedConfigurationError

exception `nidmm.errors.`**UnsupportedConfigurationError**
An error due to using this module in an usupported platform.

DriverNotInstalledError

exception `nidmm.errors.`**DriverNotInstalledError**
An error due to using this module without the driver runtime installed.

InvalidRepeatedCapabilityError

exception `nidmm.errors.`**InvalidRepeatedCapabilityError**
An error due to an invalid character in a repeated capability

SelfTestError

exception `nidmm.errors.`**SelfTestError**
An error due to a failed self-test

DriverWarning

exception `nidmm.errors.`**DriverWarning**
A warning originating from the NI-DMM driver

Examples

You can download all nidmm examples here

nidmm_fetch_waveform.py

Listing 7: (nidmm_fetch_waveform.py)

```python
#!/usr/bin/python

import argparse
import nidmm
import sys
import time


def example(resource_name, options, function, range, points, rate):
    with nidmm.Session(resource_name=resource_name, options=options) as session:
        session.configure_waveform_acquisition(measurement_function=nidmm.Function[function], range=range, rate=rate, waveform_points=points)
        with session.initiate():
            while True:
                time.sleep(0.1)
                backlog, acquisition_state = session.read_status()
                if acquisition_state == nidmm.AcquisitionStatus.FINISHED_WITH_NO_BACKLOG:
                    break
                measurements = session.fetch_waveform(array_size=backlog)
                print(measurements)


def _main(argsv):
    parser = argparse.ArgumentParser(description='Performs a waveform acquisition using the NI-DMM API.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource name of a National Instruments Digital Multimeter.')
    parser.add_argument('-f', '--function', default='WAVEFORM_VOLTAGE', choices=nidmm.Function.__members__.keys(), type=str.upper, help='Measurement function.')
    parser.add_argument('-r', '--range', default=10, type=float, help='Measurement range.')
    parser.add_argument('-p', '--points', default=10, type=int, help='Specifies the number of points to acquire before the waveform acquisition completes.')
    parser.add_argument('-s', '--rate', default=1000, type=int, help='Specifies the rate of the acquisition in samples per second.')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.function, args.range, args.points, args.rate)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '4082', 'BoardType': 'PXIe', }, }
    example('PXI1Slot2', options, 'WAVEFORM_VOLTAGE', 10, 10, 1000)


```

(continues on next page)

(continued from previous page)

```python
def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:4082; BoardType:PXIe
↪', ]
    _main(cmd_line)


if __name__ == '__main__':
    main()


```

nidmm_measurement.py

Listing 8: (nidmm_measurement.py)

```python
#!/usr/bin/python

import argparse
import nidmm
import sys


def example(resource_name, option_string, function, range, digits):
    with nidmm.Session(resource_name=resource_name, options=option_string) as session:
        session.configure_measurement_digits(measurement_function=nidmm.
↪Function[function], range=range, resolution_digits=digits)
        print(session.read())


def _main(argsv):
    supported_functions = list(nidmm.Function.__members__.keys())
    parser = argparse.ArgumentParser(description='Performs a single measurement using␣
↪the NI-DMM API.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of a National Instruments Digital Multimeter.')
    parser.add_argument('-f', '--function', default=supported_functions[0],␣
↪choices=supported_functions, type=str.upper, help='Measurement function.')
    parser.add_argument('-r', '--range', default=10, type=float, help='Measurement␣
↪range.')
    parser.add_argument('-d', '--digits', default=6.5, type=float, help='Digits of␣
↪resolution for the measurement.')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.function, args.range, args.
↪digits)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '4082', 'BoardType': 'PXIe
↪', }, }
```

(continues on next page)

(continued from previous page)

```
    example('PXI1Slot2', options, 'DC_VOLTS', 10, 6.5)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:4082; BoardType:PXIe
↪', ]
    _main(cmd_line)


if __name__ == '__main__':
    main()


```

nidmm_multi_point_measurement.py

Listing 9: (nidmm_multi_point_measurement.py)

```
#!/usr/bin/python

import argparse
import nidmm
import sys


def example(resource_name, options, function, range, digits, samples, triggers):
    with nidmm.Session(resource_name=resource_name, options=options) as session:
        session.configure_measurement_digits(measurement_function=nidmm.
↪Function[function], range=range, resolution_digits=digits)
        session.configure_multi_point(trigger_count=triggers, sample_count=samples)
        measurements = session.read_multi_point(array_size=samples)
        print('Measurements: ', measurements)


def _main(argsv):
    parser = argparse.ArgumentParser(description='Performs a multipoint measurement␣
↪using the NI-DMM API.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of a National Instruments Digital Multimeter.')
    parser.add_argument('-f', '--function', default='DC_VOLTS', choices=nidmm.
↪Function.__members__.keys(), type=str.upper, help='Measurement function.')
    parser.add_argument('-r', '--range', default=10, type=float, help='Measurement␣
↪range.')
    parser.add_argument('-d', '--digits', default=6.5, type=float, help='Digits of␣
↪resolution for the measurement.')
    parser.add_argument('-s', '--samples', default=10, type=int, help='The number of␣
↪measurements the DMM makes.')
    parser.add_argument('-t', '--triggers', default=1, type=int, help='Sets the␣
↪number of triggers you want the DMM to receive before returning to the Idle state.')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.function, args.range, args.
↪digits, args.samples, args.triggers)
```

(continues on next page)

(continued from previous page)

```


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '4082', 'BoardType': 'PXIe
↪', }, }
    example('PXI1Slot2', options, 'DC_VOLTS', 10, 6.5, 10, 1)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:4082; BoardType:PXIe
↪', ]
    _main(cmd_line)


if __name__ == '__main__':
    main()



```

7.4 nifgen module

7.4.1 Installation

As a prerequisite to using the nifgen module, you must install the NI-FGEN runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

The nimi-python modules (i.e. for **NI-FGEN**) can be installed with pip:

```
$ python -m pip install nifgen~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install nifgen
```

7.4.2 Usage

The following is a basic example of using the **nifgen** module to open a session to a Function Generator and generate a sine wave for 5 seconds.

```
import nifgen
import time
with nifgen.Session("Dev1") as session:
    session.output_mode = nifgen.OutputMode.FUNC
    session.configure_standard_waveform(waveform=nifgen.Waveform.SINE, amplitude=1.0,␣
↪frequency=10000000, dc_offset=0.0, start_phase=0.0)
    with session.initiate():
        time.sleep(5)
```

Additional examples for NI-FGEN are located in src/nifgen/examples/ directory.

7.4.3 API Reference

Session

class nifgen.**Session**(*self*, *resource_name*, *channel_name=None*, *reset_device=False*, *options={}*)

Creates and returns a new NI-FGEN session to the specified channel of a waveform generator that is used in all subsequent NI-FGEN 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 ensure 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	Variable
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)

- **channel_name** (*str, list, range, tuple*) – Specifies the channel that this VI uses.

  Default Value: "0"

- **reset_device** (*bool*) – Specifies whether you want to reset the device during the initialization procedure. True specifies that the device is reset and performs the same method as the `nifgen.Session.Reset()` method.

  Defined Values

  Default Value: False

True	Reset device
False	Do not reset device

- **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	{}

Methods

abort

`nifgen.Session.abort()`

Aborts any previously initiated signal generation. Call the `nifgen.Session.initiate()`

method to cause the signal generator to produce a signal again.

allocate_named_waveform

`nifgen.Session.`**`allocate_named_waveform`**`(`*waveform_name*, *waveform_size*`)`

Specifies the size of a named waveform up front so that it can be allocated in onboard memory before loading the associated data. Data can then be loaded in smaller blocks with the niFgen Write (Binary16) Waveform methods.

Tip: This method can be called on specific channels within your `nifgen.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[ ... ].allocate_named_waveform()`

To call the method on all channels, you can call it directly on the `nifgen.Session`.

Example: `my_session.allocate_named_waveform()`

Parameters

- **`waveform_name`** (*`str`*) – Specifies the name to associate with the allocated waveform.
- **`waveform_size`** (*`int`*) – Specifies the size of the waveform to allocate in samples.

  Default Value: “4096”

allocate_waveform

`nifgen.Session.`**`allocate_waveform`**`(`*waveform_size*`)`

Specifies the size of a waveform so that it can be allocated in onboard memory before loading the associated data. Data can then be loaded in smaller blocks with the Write Binary 16 Waveform methods.

Note: The signal generator must not be in the Generating state when you call this method.

Tip: This method can be called on specific channels within your `nifgen.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[ ... ].allocate_waveform()`

To call the method on all channels, you can call it directly on the `nifgen.Session`.

Example: `my_session.allocate_waveform()`

Parameters **`waveform_size`** (*`int`*) – Specifies, in samples, the size of the waveform to allocate.

Return type int

Returns The handle that identifies the new waveform. This handle is used later when referring to this waveform.

clear_arb_memory

`nifgen.Session.`**`clear_arb_memory`**`()`

Removes all previously created arbitrary waveforms, sequences, and scripts from the signal generator memory and invalidates all waveform handles, sequence handles, and waveform names.

Note: The signal generator must not be in the Generating state when you call this method.

clear_arb_sequence

`nifgen.Session.`**`clear_arb_sequence`**`(`*sequence_handle*`)`

Removes a previously created arbitrary sequence from the signal generator memory and invalidates the sequence handle.

Note: The signal generator must not be in the Generating state when you call this method.

Parameters **`sequence_handle`** (*`int`*) – Specifies the handle of the arbitrary sequence that you want the signal generator to remove. You can create an arbitrary sequence using the *`nifgen.Session.create_arb_sequence()`* or *`nifgen.Session.create_advanced_arb_sequence()`* method. These methods return a handle that you use to identify the sequence.

Defined Value:

`NIFGEN_VAL_ALL_SEQUENCES`—Remove all sequences from the signal generator

Default Value: None

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.

clear_freq_list

`nifgen.Session.`**`clear_freq_list`**`(`*frequency_list_handle*`)`

Removes a previously created frequency list from the signal generator memory and invalidates the frequency list handle.

Note: The signal generator must not be in the Generating state when you call this method.

Parameters **`frequency_list_handle`** (*int*) – Specifies the handle of the frequency list you want the signal generator to remove. You create multiple frequency lists using *`nifgen.Session.create_freq_list()`*. *`nifgen.Session.create_freq_list()`* returns a handle that you use to identify each list. Specify a value of -1 to clear all frequency lists.

Defined Value

`NIFGEN_VAL_ALL_FLISTS`—Remove all frequency lists from the signal generator.

Default Value: None

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.

clear_user_standard_waveform

`nifgen.Session.`**`clear_user_standard_waveform`**`()`

Clears the user-defined waveform created by the *`nifgen.Session.define_user_standard_waveform()`* method.

Tip: This method can be called on specific channels within your *`nifgen.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_user_standard_waveform()`

To call the method on all channels, you can call it directly on the *`nifgen.Session`*.

Example: `my_session.clear_user_standard_waveform()`

close

`nifgen.Session.`**`close`**`()`

Performs the following operations:

- Closes the instrument I/O session.
- Destroys the NI-FGEN session and all of its properties.
- Deallocates any memory resources NI-FGEN uses.

Not all signal routes established by calling the `nifgen.Session.ExportSignal()` and `nifgen.Session.RouteSignalOut()` methods are released when the NI-FGEN session is closed. The following table shows what happens to a signal route on your device when you call the `nifgen.Session._close()` method.

Routes To	NI 5401/5411/5431	Other Devices
Front Panel	Remain connected	Remain connected
RTSI/PXI Backplane	Remain connected	Disconnected

Note: After calling `nifgen.Session._close()`, you cannot use NI-FGEN again until you call the `nifgen.Session.init()` or `nifgen.Session.InitWithOptions()` methods.

Note: This method is not needed when using the session context manager

commit

`nifgen.Session.`**`commit`**`()`

Causes a transition to the Committed state. This method verifies property values, reserves the device, and commits the property values to the device. If the property values are all valid, NI-FGEN sets the device hardware configuration to match the session configuration. This method does not support the NI 5401/5404/5411/5431 signal generators.

In the Committed state, you can load waveforms, scripts, and sequences into memory. If any properties are changed, NI-FGEN implicitly transitions back to the Idle state, where you can program all session properties before applying them to the device. This method has no effect if the device is already in the Committed or Generating state and returns a successful status value.

Calling this VI before the niFgen Initiate Generation VI is optional but has the following benefits:

- Routes are committed, so signals are exported or imported.
- Any Reference Clock and external clock circuits are phase-locked.
- A subsequent `nifgen.Session.initiate()` method can run faster because the device is already configured.

configure_arb_sequence

`nifgen.Session.`**`configure_arb_sequence`**`(`*sequence_handle*, *gain*, *offset*`)`

Configures the signal generator properties that affect arbitrary sequence generation. Sets the `nifgen.Session.arb_sequence_handle`, `nifgen.Session.arb_gain`, and `nifgen.Session.arb_offset` properties.

Note: The signal generator must not be in the Generating state when you call this method.

Tip: This method can be called on specific channels within your `nifgen.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_arb_sequence()`

To call the method on all channels, you can call it directly on the `nifgen.Session`.

Example: `my_session.configure_arb_sequence()`

Parameters

- **sequence_handle** (*int*) – Specifies the handle of the arbitrary sequence that you want the signal generator to produce. NI-FGEN sets the `nifgen.Session.arb_sequence_handle` property to this value. You can create an arbitrary sequence using the `nifgen.Session.create_arb_sequence()` or `nifgen.Session.create_advanced_arb_sequence()` method. These methods return a handle that you use to identify the sequence.

  Default Value: None

- **gain** (*float*) – Specifies the factor by which the signal generator scales the arbitrary waveforms in the sequence. When you create an arbitrary waveform, you must first normalize the data points to a range of –1.00 to +1.00. You can use this parameter to scale the waveform to other ranges. The gain is applied before the offset is added.

  For example, to configure the output signal to range from –2.00 to +2.00 V, set **gain** to 2.00.

  Units: unitless

  Default Value: None

- **offset** (*float*) – Specifies the value the signal generator adds to the arbitrary waveform data. When you create arbitrary waveforms, you must first normalize the data points to a range of –1.00 to +1.00 V. You can use this parameter to shift the range of the arbitrary waveform. NI-FGEN sets the `nifgen.Session.arb_offset` property to this value.

  For example, to configure the output signal to range from 0.00 to 2.00 V instead of –1.00 to 1.00 V, set the offset to 1.00.

  Units: volts

  Default Value: None

configure_arb_waveform

`nifgen.Session.`**`configure_arb_waveform`**`(`*waveform_handle*, *gain*, *offset*`)`

Configures the properties of the signal generator that affect arbitrary waveform generation. Sets the `nifgen.Session.arb_waveform_handle`, `nifgen.Session.arb_gain`, and `nifgen.Session.arb_offset` properties.

Note: The signal generator must not be in the Generating state when you call this method.

Tip: This method can be called on specific channels within your `nifgen.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_arb_waveform()`

To call the method on all channels, you can call it directly on the `nifgen.Session`.

Example: `my_session.configure_arb_waveform()`

Parameters

- **waveform_handle** (*int*) – Specifies the handle of the arbitrary waveform you want the signal generator to produce. NI-FGEN sets the *nifgen.Session.arb_waveform_handle* property to this value. You can create an arbitrary waveform using one of the following niFgen Create Waveform methods:
  - nifgen.Session.create_waveform()
  - nifgen.Session.create_waveform()
  - *nifgen.Session.create_waveform_from_file_i16()*
  - *nifgen.Session.create_waveform_from_file_f64()*
  - nifgen.Session.CreateWaveformFromFileHWS()

  These methods return a handle that you use to identify the waveform.

  Default Value: None

  Note: One or more of the referenced methods are not in the Python API for this driver.

- **gain** (*float*) – Specifies the factor by which the signal generator scales the arbitrary waveforms in the sequence. When you create an arbitrary waveform, you must first normalize the data points to a range of –1.00 to +1.00. You can use this parameter to scale the waveform to other ranges. The gain is applied before the offset is added.

  For example, to configure the output signal to range from –2.00 to +2.00 V, set **gain** to 2.00.

  Units: unitless

  Default Value: None

- **offset** (*float*) – Specifies the value the signal generator adds to the arbitrary waveform data. When you create arbitrary waveforms, you must first normalize the data points to a range of –1.00 to +1.00 V. You can use this parameter to shift the range of the arbitrary waveform. NI-FGEN sets the *nifgen.Session.arb_offset* property to this value.

  For example, to configure the output signal to range from 0.00 to 2.00 V instead of –1.00 to 1.00 V, set the offset to 1.00.

  Units: volts

  Default Value: None

configure_freq_list

nifgen.Session.**configure_freq_list**(*frequency_list_handle*, *amplitude*, *dc_offset=0.0*, *start_phase=0.0*)

Configures the properties of the signal generator that affect frequency list generation (the *nifgen.Session.freq_list_handle*, *nifgen.Session.func_amplitude*, *nifgen.Session.func_dc_offset*, and *nifgen.Session.func_start_phase* properties).

Note: The signal generator must not be in the Generating state when you call this method.

Tip: This method can be called on specific channels within your *nifgen.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_freq_list()`

To call the method on all channels, you can call it directly on the *nifgen.Session*.

Example: `my_session.configure_freq_list()`

Parameters

- **frequency_list_handle** (*int*) – Specifies the handle of the frequency list that you want the signal generator to produce. NI-FGEN sets the *nifgen.Session.freq_list_handle* property to this value. You can create a frequency list using the *nifgen.Session.create_freq_list()* method, which returns a handle that you use to identify the list. **Default Value**: None
- **amplitude** (*float*) – Specifies the amplitude of the standard waveform that you want the signal generator to produce. This value is the amplitude at the output terminal. NI-FGEN sets the *nifgen.Session.func_amplitude* property to this value.

  For example, to produce a waveform ranging from –5.00 V to +5.00 V, set the amplitude to 10.00 V.

  Units: peak-to-peak voltage

  Default Value: None

  Note: This parameter does not affect signal generator behavior when you set the **waveform** parameter of the *nifgen.Session.configure_standard_waveform()* method to *DC*.

- **dc_offset** (*float*) – Specifies the DC offset of the standard waveform that you want the signal generator to produce. The value is the offset from ground to the center of the waveform you specify with the **waveform** parameter, observed at the output terminal. For example, to configure a waveform with an amplitude of 10.00 V to range from 0.00 V to +10.00 V, set the **dcOffset** to 5.00 V. NI-FGEN sets the *nifgen.Session.func_dc_offset* property to this value.

  Units: volts

  Default Value: None

- **start_phase** (*float*) – Specifies the horizontal offset of the standard waveform you want the signal generator to produce. Specify this property in degrees of one waveform cycle. NI-FGEN sets the *nifgen.Session.func_start_phase* property to this value. A start phase of 180 degrees means output generation begins halfway through the waveform. A start phase of 360 degrees offsets the output by an entire waveform cycle, which is identical to a start phase of 0 degrees.

  Units: degrees of one cycle

  Default Value: None degrees

Note: This parameter does not affect signal generator behavior when you set the **waveform** parameter to *DC*.

configure_standard_waveform

`nifgen.Session.`**`configure_standard_waveform`**`(`*waveform*, *amplitude*, *frequency*, *dc_offset=0.0*, *start_phase=0.0*`)`

Configures the following properties of the signal generator that affect standard waveform generation:

- *nifgen.Session.func_waveform*
- *nifgen.Session.func_amplitude*
- *nifgen.Session.func_dc_offset*
- *nifgen.Session.func_frequency*
- *nifgen.Session.func_start_phase*

Note: You must call the `nifgen.Session.ConfigureOutputMode()` method with the **outputMode** parameter set to *FUNC* before calling this method.

Note: One or more of the referenced methods are not in the Python API for this driver.

Tip: This method can be called on specific channels within your *nifgen.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_standard_waveform()`

To call the method on all channels, you can call it directly on the *nifgen.Session*.

Example: `my_session.configure_standard_waveform()`

Parameters

- **`waveform`** (*nifgen.Waveform*) – Specifies the standard waveform that you want the signal generator to produce. NI-FGEN sets the *nifgen.Session.func_waveform* property to this value.

  ****Defined Values****

  Default Value: *SINE*

SINE	Specifies that the signal generator produces a sinusoid waveform.
SQUARE	Specifies that the signal generator produces a square waveform.
TRIANGLE	Specifies that the signal generator produces a triangle waveform.
RAMP_UP	Specifies that the signal generator produces a positive ramp waveform.
RAMP_DOWN	Specifies that the signal generator produces a negative ramp waveform.
DC	Specifies that the signal generator produces a constant voltage.
NOISE	Specifies that the signal generator produces white noise.
USER	Specifies that the signal generator produces a user-defined waveform as defined with the `nifgen.Session.define_user_standard_waveform()` method.

- **amplitude** (*float*) – Specifies the amplitude of the standard waveform that you want the signal generator to produce. This value is the amplitude at the output terminal. NI-FGEN sets the `nifgen.Session.func_amplitude` property to this value.

  For example, to produce a waveform ranging from –5.00 V to +5.00 V, set the amplitude to 10.00 V.

  Units: peak-to-peak voltage

  Default Value: None

  Note: This parameter does not affect signal generator behavior when you set the **waveform** parameter of the `nifgen.Session.configure_standard_waveform()` method to *DC*.

- **frequency** (*float*) –

  Specifies the frequency of the standard waveform that you want the signal generator to produce. NI-FGEN sets the `nifgen.Session.func_frequency` property to this value.

  Units: hertz

  Default Value: None

  Note: This parameter does not affect signal generator behavior when you set the **waveform** parameter of the `nifgen.Session.configure_standard_waveform()` method to *DC*.

- **dc_offset** (*float*) – Specifies the DC offset of the standard waveform that you want the signal generator to produce. The value is the offset from ground to the center of the waveform you specify with the **waveform** parameter, observed at the output terminal. For example, to configure a waveform with an amplitude of 10.00 V to range from 0.00 V to +10.00 V, set the **dcOffset** to 5.00 V. NI-FGEN sets the `nifgen.Session.func_dc_offset` property to this value.

  Units: volts

  Default Value: None

- **start_phase** (*float*) – Specifies the horizontal offset of the standard waveform that you want the signal generator to produce. Specify this parameter in degrees of one waveform cycle. NI-FGEN sets the *nifgen.Session.func_start_phase* property to this value. A start phase of 180 degrees means output generation begins halfway through the waveform. A start phase of 360 degrees offsets the output by an entire waveform cycle, which is identical to a start phase of 0 degrees.

Units: degrees of one cycle

Default Value: 0.00

Note: This parameter does not affect signal generator behavior when you set the **waveform** parameter to *DC*.

create_advanced_arb_sequence

`nifgen.Session.`**`create_advanced_arb_sequence`**`(`*waveform_handles_array*, *loop_counts_array*, *sample_counts_array=None*, *marker_location_array=None*`)`

Creates an arbitrary sequence from an array of waveform handles and an array of corresponding loop counts. This method returns a handle that identifies the sequence. You pass this handle to the *nifgen.Session.configure_arb_sequence()* method to specify what arbitrary sequence you want the signal generator to produce.

The *nifgen.Session.create_advanced_arb_sequence()* method extends on the *nifgen.Session.create_arb_sequence()* method by adding the ability to set the number of samples in each sequence step and to set marker locations.

An arbitrary sequence consists of multiple waveforms. For each waveform, you specify the number of times the signal generator produces the waveform before proceeding to the next waveform. The number of times to repeat a specific waveform is called the loop count.

Note: The signal generator must not be in the Generating state when you call this method. You must call the `nifgen.Session.ConfigureOutputMode()` method to set the **outputMode** parameter to *SEQ* before calling this method.

Parameters

- **`waveform_handles_array`** (*list of int*) – Specifies the array of waveform handles from which you want to create a new arbitrary sequence. The array must have at least as many elements as the value that you specify in **sequenceLength**. Each **waveformHandlesArray** element has a corresponding **loopCountsArray** element that indicates how many times that waveform is repeated. You obtain waveform handles when you create arbitrary waveforms with the *nifgen.Session.allocate_waveform()* method or one of the following niFgen CreateWaveform methods:
  - `nifgen.Session.create_waveform()`
  - `nifgen.Session.create_waveform()`
  - *nifgen.Session.create_waveform_from_file_i16()*
  - *nifgen.Session.create_waveform_from_file_f64()*

– `nifgen.Session.CreateWaveformFromFileHWS()`

Default Value: None

- **`loop_counts_array`** (*`list of int`*) – Specifies the array of loop counts you want to use to create a new arbitrary sequence. The array must have at least as many elements as the value that you specify in the **sequenceLength** parameter. Each **loopCountsArray** element corresponds to a **waveformHandlesArray** element and indicates how many times to repeat that waveform. Each element of the **loopCountsArray** must be less than or equal to the maximum number of loop counts that the signal generator allows. You can obtain the maximum loop count from **maximumLoopCount** in the *`nifgen.Session.query_arb_seq_capabilities()`* method.

  Default Value: None

- **`sample_counts_array`** (*`list of int`*) – Specifies the array of sample counts that you want to use to create a new arbitrary sequence. The array must have at least as many elements as the value you specify in the **sequenceLength** parameter. Each **sampleCountsArray** element corresponds to a **waveformHandlesArray** element and indicates the subset, in samples, of the given waveform to generate. Each element of the **sampleCountsArray** must be larger than the minimum waveform size, a multiple of the waveform quantum and no larger than the number of samples in the corresponding waveform. You can obtain these values by calling the *`nifgen.Session.query_arb_wfm_capabilities()`* method.

  Default Value: None

- **`marker_location_array`** (*`list of int`*) – Specifies the array of marker locations to where you want a marker to be generated in the sequence. The array must have at least as many elements as the value you specify in the **sequenceLength** parameter. Each **markerLocationArray** element corresponds to a **waveformHandlesArray** element and indicates where in the waveform a marker is to generate. The marker location must be less than the size of the waveform the marker is in. The markers are coerced to the nearest marker quantum and the coerced values are returned in the **coercedMarkersArray** parameter.

  If you do not want a marker generated for a particular sequence stage, set this parameter to `NIFGEN_VAL_NO_MARKER`.

  Defined Value: `NIFGEN_VAL_NO_MARKER`

  Default Value: None

  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.

Return type

tuple (coerced_markers_array, sequence_handle)

WHERE

coerced_markers_array (list of int):

Returns an array of all given markers that are coerced (rounded) to the nearest marker quantum. Not all devices coerce markers.

Default Value: None

sequence_handle (int):

Returns the handle that identifies the new arbitrary sequence. You can pass this handle to `nifgen.Session.configure_arb_sequence()` to generate the arbitrary sequence.

create_arb_sequence

`nifgen.Session.`**`create_arb_sequence`**`(`*waveform_handles_array*, *loop_counts_array*`)`

Creates an arbitrary sequence from an array of waveform handles and an array of corresponding loop counts. This method returns a handle that identifies the sequence. You pass this handle to the `nifgen.Session.configure_arb_sequence()` method to specify what arbitrary sequence you want the signal generator to produce.

An arbitrary sequence consists of multiple waveforms. For each waveform, you can specify the number of times that the signal generator produces the waveform before proceeding to the next waveform. The number of times to repeat a specific waveform is called the loop count.

Note: You must call the `nifgen.Session.ConfigureOutputMode()` method to set the **outputMode** parameter to `SEQ` before calling this method.

Parameters

- **`waveform_handles_array`** (*`list of int`*) – Specifies the array of waveform handles from which you want to create a new arbitrary sequence. The array must have at least as many elements as the value that you specify in **sequenceLength**. Each **waveformHandlesArray** element has a corresponding **loopCountsArray** element that indicates how many times that waveform is repeated. You obtain waveform handles when you create arbitrary waveforms with the `nifgen.Session.allocate_waveform()` method or one of the following niFgen CreateWaveform methods:
  - `nifgen.Session.create_waveform()`
  - `nifgen.Session.create_waveform()`
  - `nifgen.Session.create_waveform_from_file_i16()`
  - `nifgen.Session.create_waveform_from_file_f64()`
  - `nifgen.Session.CreateWaveformFromFileHWS()`

  Default Value: None

- **`loop_counts_array`** (*`list of int`*) – Specifies the array of loop counts you want to use to create a new arbitrary sequence. The array must have at least as many elements as the value that you specify in the **sequenceLength** parameter. Each **loopCountsArray** element corresponds to a **waveformHandlesArray** element and indicates how many times to repeat that waveform. Each element of the **loopCountsArray** must be less than or equal to the maximum number of loop counts that the signal generator allows. You can obtain the maximum loop count from **maximumLoopCount** in the `nifgen.Session.query_arb_seq_capabilities()` method.

  Default Value: None

Return type int

Returns Returns the handle that identifies the new arbitrary sequence. You can pass this handle to *nifgen.Session.configure_arb_sequence()* to generate the arbitrary sequence.

create_freq_list

nifgen.Session.**create_freq_list**(*waveform*, *frequency_array*, *duration_array*)

Creates a frequency list from an array of frequencies (**frequencyArray**) and an array of durations (**durationArray**). The two arrays should have the same number of elements, and this value must also be the size of the **frequencyListLength**. The method returns a handle that identifies the frequency list (the **frequencyListHandle**). You can pass this handle to *nifgen.Session.configure_freq_list()* to specify what frequency list you want the signal generator to produce.

A frequency list consists of a list of frequencies and durations. The signal generator generates each frequency for the given amount of time and then proceeds to the next frequency. When the end of the list is reached, the signal generator starts over at the beginning of the list.

Note: The signal generator must not be in the Generating state when you call this method.

Parameters

- **waveform** (*nifgen.Waveform*) – Specifies the standard waveform that you want the signal generator to produce. NI-FGEN sets the *nifgen.Session.func_waveform* property to this value.

  ****Defined Values****

  Default Value: *SINE*

SINE	Specifies that the signal generator produces a sinusoid waveform.
SQUARE	Specifies that the signal generator produces a square waveform.
TRIANGLE	Specifies that the signal generator produces a triangle waveform.
RAMP_UP	Specifies that the signal generator produces a positive ramp waveform.
RAMP_DOWN	Specifies that the signal generator produces a negative ramp waveform.
DC	Specifies that the signal generator produces a constant voltage.
NOISE	Specifies that the signal generator produces white noise.
USER	Specifies that the signal generator produces a user-defined waveform as defined with the *nifgen.Session.define_user_standard_waveform()* method.

- **frequency_array** (*list of float*) – Specifies the array of frequencies to form the frequency list. The array must have at least as many elements as the value you specify in **frequencyListLength**. Each **frequencyArray** element has a corresponding **durationArray** element that indicates how long that frequency is repeated.

  Units: hertz

  Default Value: None

- **duration_array** (*list of float*) – Specifies the array of durations to form the frequency list. The array must have at least as many elements as the value that you specify in **frequencyListLength**. Each **durationArray** element has a corresponding **frequencyArray** element and indicates how long in seconds to generate the corresponding frequency.

Units: seconds

Default Value: None

Return type int

Returns Returns the handle that identifies the new frequency list. You can pass this handle to *nifgen.Session.configure_freq_list()* to generate the arbitrary sequence.

create_waveform_from_file_f64

nifgen.Session.**create_waveform_from_file_f64**(*file_name*, *byte_order*)

This method takes the floating point double (F64) data from the specified file and creates an onboard waveform for use in Arbitrary Waveform or Arbitrary Sequence output mode. The **waveformHandle** returned by this method can later be used for setting the active waveform, changing the data in the waveform, building sequences of waveforms, or deleting the waveform when it is no longer needed.

Note: The F64 data must be between –1.0 and +1.0 V. Use the *nifgen.Session.digital_gain* property to generate different voltage outputs.

Tip: This method can be called on specific channels within your *nifgen.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[ ... ].create_waveform_from_file_f64()`

To call the method on all channels, you can call it directly on the *nifgen.Session*.

Example: `my_session.create_waveform_from_file_f64()`

Parameters

- **file_name** (*str*) – The full path and name of the file where the waveform data resides.
- **byte_order** (*nifgen.ByteOrder*) – Specifies the byte order of the data in the file.

  ****Defined Values****

  ****Default Value**:** *LITTLE*

LITTLE	Little Endian Data—The least significant bit is stored at the lowest address, followed by the other bits, in order of increasing significance.
BIG	Big Endian Data—The most significant bit is stored at the lowest address, followed by the other bits, in order of decreasing significance.

Note: Data written by most applications in Windows (including LabWindows™/CVI™) is in Little Endian format. Data written to a file from LabVIEW is in Big Endian format by default on all platforms. Big Endian and Little Endian refer to the way data is stored in memory, which can differ on different processors.

Return type int

Returns The handle that identifies the new waveform. This handle is used later when referring to this waveform.

create_waveform_from_file_i16

`nifgen.Session.`**`create_waveform_from_file_i16`**(*file_name*, *byte_order*)

Takes the binary 16-bit signed integer (I16) data from the specified file and creates an onboard waveform for use in Arbitrary Waveform or Arbitrary Sequence output mode. The **waveformHandle** returned by this method can later be used for setting the active waveform, changing the data in the waveform, building sequences of waveforms, or deleting the waveform when it is no longer needed.

Note: The I16 data (values between –32768 and +32767) is assumed to represent –1 to +1 V. Use the *nifgen.Session.digital_gain* property to generate different voltage outputs.

Tip: This method can be called on specific channels within your *nifgen.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[ ... ].create_waveform_from_file_i16()`

To call the method on all channels, you can call it directly on the *nifgen.Session*.

Example: `my_session.create_waveform_from_file_i16()`

Parameters

- **`file_name`** (*str*) – The full path and name of the file where the waveform data resides.
- **`byte_order`** (*nifgen.ByteOrder*) – Specifies the byte order of the data in the file.

  ****Defined Values****

  ****Default Value**:** *LITTLE*

LITTLE	Little Endian Data—The least significant bit is stored at the lowest address, followed by the other bits, in order of increasing significance.
BIG	Big Endian Data—The most significant bit is stored at the lowest address, followed by the other bits, in order of decreasing significance.

Note: Data written by most applications in Windows (including LabWindows™/CVI™) is in Little Endian format. Data written to a file from LabVIEW is in Big Endian format by default on all platforms. Big Endian and Little Endian refer to the way data is stored in memory, which can differ on different processors.

Return type int

Returns The handle that identifies the new waveform. This handle is used later when referring to this waveform.

create_waveform_numpy

nifgen.Session.**create_waveform_numpy**(*waveform_data_array*)

Creates an onboard waveform for use in Arbitrary Waveform output mode or Arbitrary Sequence output mode.

Note: You must set *nifgen.Session.output_mode* to *ARB* or *SEQ* before calling this method.

Tip: This method can be called on specific channels within your *nifgen.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[ ... ].create_waveform()`

To call the method on all channels, you can call it directly on the *nifgen.Session*.

Example: `my_session.create_waveform()`

Parameters **waveform_data_array** (*iterable of float or int16*) – Array of data for the new arbitrary waveform. This may be an iterable of float or int16, or for best performance a numpy.ndarray of dtype int16 or float64.

Return type int

Returns The handle that identifies the new waveform. This handle is used in other methods when referring to this waveform.

define_user_standard_waveform

nifgen.Session.**define_user_standard_waveform**(*waveform_data_array*)

Defines a user waveform for use in either Standard Method or Frequency List output mode.

To select the waveform, set the **waveform** parameter to *USER* with either the *nifgen.Session.configure_standard_waveform()* or the *nifgen.Session.create_freq_list()* method.

The waveform data must be scaled between –1.0 and 1.0. Use the **amplitude** parameter in the *nifgen.Session.configure_standard_waveform()* method to generate different output voltages.

Note: You must call the `nifgen.Session.ConfigureOutputMode()` method to set the **outputMode** parameter to *FUNC* or *FREQ_LIST* before calling this method.

Tip: This method can be called on specific channels within your *nifgen.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[ ... ].define_user_standard_waveform()`

To call the method on all channels, you can call it directly on the *nifgen.Session*.

Example: `my_session.define_user_standard_waveform()`

Parameters **waveform_data_array** (*list of float*) – Specifies the array of data you want to use for the new arbitrary waveform. The array must have at least as many elements as the value that you specify in **waveformSize**.

You must normalize the data points in the array to be between –1.00 and +1.00.

Default Value: None

delete_script

`nifgen.Session.`**`delete_script`**`(`*script_name*`)`

Deletes the specified script from onboard memory.

Tip: This method can be called on specific channels within your *nifgen.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[ ... ].delete_script()`

To call the method on all channels, you can call it directly on the *nifgen.Session*.

Example: `my_session.delete_script()`

Parameters **script_name** (*str*) – Specifies the name of the script you want to delete. The script name appears in the text of the script following the script keyword.

delete_waveform

`nifgen.Session.`**`delete_waveform`**`(`*waveform_name_or_handle*`)`

Removes a previously created arbitrary waveform from the signal generator memory.

Note: The signal generator must not be in the Generating state when you call this method.

Tip: This method can be called on specific channels within your *nifgen.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[ ... ].delete_waveform()`

To call the method on all channels, you can call it directly on the *nifgen.Session*.

Example: `my_session.delete_waveform()`

Parameters **waveform_name_or_handle** (*str or int*) – The name (str) or handle (int) of an arbitrary waveform previously allocated with *nifgen.Session.allocate_named_waveform()*, *nifgen.Session.allocate_waveform()* or `nifgen.Session.create_waveform()`.

disable

`nifgen.Session.`**`disable`**`()`

Places the instrument in a quiescent state where it has minimal or no impact on the system to which it is connected. The analog output and all exported signals are disabled.

export_attribute_configuration_buffer

`nifgen.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-FGEN returns an error.

Return type bytes

Returns Specifies the byte array buffer to be populated with the exported property configuration.

export_attribute_configuration_file

`nifgen.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-FGEN returns an error.

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:** .nifgenconfig

get_channel_name

nifgen.Session.**get_channel_name**(*index*)

Returns the channel string that is in the channel table at an index you specify.

Note: This method is included for compliance with the IviFgen Class Specification.

Parameters **index** (*int*) – A 1-based index into the channel table.

Return type str

Returns Returns the channel string that is in the channel table at the index you specify. Do not modify the contents of the channel string.

get_ext_cal_last_date_and_time

nifgen.Session.**get_ext_cal_last_date_and_time**()

Returns the date and time of the last successful external calibration. The time returned is 24-hour (military) local time; for example, if the device was calibrated at 2:30 PM, this method returns 14 for the **hour** parameter and 30 for the **minute** parameter.

Return type hightime.datetime

Returns Indicates date and time of the last calibration.

get_ext_cal_last_temp

nifgen.Session.**get_ext_cal_last_temp**()

Returns the temperature at the last successful external calibration. The temperature is returned in degrees Celsius.

Return type float

Returns Specifies the temperature at the last successful calibration in degrees Celsius.

get_ext_cal_recommended_interval

nifgen.Session.**get_ext_cal_recommended_interval**()

Returns the recommended interval between external calibrations in months.

Return type hightime.timedelta

Returns Specifies the recommended interval between external calibrations in months.

get_hardware_state

nifgen.Session.**get_hardware_state**()

Returns the current hardware state of the device and, if the device is in the hardware error state, the current hardware error.

Note: Hardware states do not necessarily correspond to NI-FGEN states.

Return type *nifgen.HardwareState*

Returns

Returns the hardware state of the signal generator.

Defined Values

IDLE	The device is in the Idle state.
WAITING_FOR_START_TRIGGER	The device is waiting for Start Trigger.
RUNNING	The device is in the Running state.
DONE	The generation has completed successfully.
HARDWARE_ERROR	There is a hardware error.

get_self_cal_last_date_and_time

`nifgen.Session.`**`get_self_cal_last_date_and_time`**`()`

Returns the date and time of the last successful self-calibration.

Return type hightime.datetime

Returns Returns the date and time the device was last calibrated.

get_self_cal_last_temp

`nifgen.Session.`**`get_self_cal_last_temp`**`()`

Returns the temperature at the last successful self-calibration. The temperature is returned in degrees Celsius.

Return type float

Returns Specifies the temperature at the last successful calibration in degrees Celsius.

get_self_cal_supported

`nifgen.Session.`**`get_self_cal_supported`**`()`

Returns whether the device supports self–calibration.

Return type bool

Returns

Returns whether the device supports self-calibration.

****Defined Values****

True	Self–calibration is supported.
False	Self–calibration is not supported.

import_attribute_configuration_buffer

`nifgen.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.

Note: You cannot call this method while the session is in a running state, such as while generating a signal.

Parameters **`configuration`** (*`bytes`*) – Specifies the byte array buffer that contains the property configuration to import.

import_attribute_configuration_file

`nifgen.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.

Note: You cannot call this method while the session is in a running state, such as while generating 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:** .nifgenconfig

initiate

`nifgen.Session.`**`initiate`**`()`

Initiates signal generation. If you want to abort signal generation, call the *`nifgen.Session.abort()`* method. After the signal generation is aborted, you can call the *`nifgen.Session.initiate()`* method to cause the signal generator to produce a signal again.

Note: This method will return a Python context manager that will initiate on entering and abort on exit.

is_done

`nifgen.Session.`**`is_done`**`()`

Determines whether the current generation is complete. This method sets the **done** parameter to True if the session is in the Idle or Committed states.

Note: NI-FGEN only reports the **done** parameter as True after the current generation is complete in Single trigger mode.

Return type bool

Returns

Returns information about the completion of waveform generation.

Defined Values

True	Generation is complete.
False	Generation is not complete.

lock

`nifgen.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 `nifgen.Session.lock()` method.
- A call to NI-FGEN locked the session.
- After a call to the `nifgen.Session.lock()` method returns successfully, no other threads can access the device session until you call the `nifgen.Session.unlock()` method or exit out of the with block when using lock context manager.
- Use the `nifgen.Session.lock()` method and the `nifgen.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 `nifgen.Session.lock()` method within the same thread. To completely unlock the session, you must balance each call to the `nifgen.Session.lock()` method with a call to the `nifgen.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 nifgen.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, `nifgen.Session.lock()` acts as a context manager and unlock will be called when the *with* block is exited

query_arb_seq_capabilities

`nifgen.Session.query_arb_seq_capabilities()`

Returns the properties of the signal generator that are related to creating arbitrary sequences (the `nifgen.Session.max_num_sequences`, `nifgen.Session.min_sequence_length`, `nifgen.Session.max_sequence_length`, and `nifgen.Session.max_loop_count` properties).

Return type

tuple (maximum_number_of_sequences, minimum_sequence_length, maximum_sequence_length, maximum_loop_count)

WHERE

maximum_number_of_sequences (int):

Returns the maximum number of arbitrary waveform sequences that the signal generator allows. NI-FGEN obtains this value from the *nifgen.Session.max_num_sequences* property.

minimum_sequence_length (int):

Returns the minimum number of arbitrary waveforms the signal generator allows in a sequence. NI-FGEN obtains this value from the *nifgen.Session.min_sequence_length* property.

maximum_sequence_length (int):

Returns the maximum number of arbitrary waveforms the signal generator allows in a sequence. NI-FGEN obtains this value from the *nifgen.Session.max_sequence_length* property.

maximum_loop_count (int):

Returns the maximum number of times the signal generator can repeat an arbitrary waveform in a sequence. NI-FGEN obtains this value from the *nifgen.Session.max_loop_count* property.

query_arb_wfm_capabilities

nifgen.Session.**query_arb_wfm_capabilities**()

Returns the properties of the signal generator that are related to creating arbitrary waveforms. These properties are the maximum number of waveforms, waveform quantum, minimum waveform size, and maximum waveform size.

Note: If you do not want to obtain the waveform quantum, pass a value of VI_NULL for this parameter.

Return type

tuple (maximum_number_of_waveforms, waveform_quantum, minimum_waveform_size, maximum_waveform_size)

WHERE

maximum_number_of_waveforms (int):

Returns the maximum number of arbitrary waveforms that the signal generator allows. NI-FGEN obtains this value from the *nifgen.Session.max_num_waveforms* property.

waveform_quantum (int):

The size (number of points) of each waveform must be a multiple of a constant quantum value. This parameter obtains the quantum value that the signal generator uses. NI-FGEN returns this value from the *nifgen.Session.waveform_quantum* property.

For example, when this property returns a value of 8, all waveform sizes must be a multiple of 8.

minimum_waveform_size (int):

Returns the minimum number of points that the signal generator allows in a waveform. NI-FGEN obtains this value from the *nifgen.Session.min_waveform_size* property.

maximum_waveform_size (int):

Returns the maximum number of points that the signal generator allows in a waveform. NI-FGEN obtains this value from the *nifgen.Session.max_waveform_size* property.

query_freq_list_capabilities

`nifgen.Session.`**`query_freq_list_capabilities`**`()`

Returns the properties of the signal generator that are related to creating frequency lists. These properties are *nifgen.Session.max_num_freq_lists*, *nifgen.Session.min_freq_list_length*, *nifgen.Session.max_freq_list_length*, *nifgen.Session.min_freq_list_duration*, *nifgen.Session.max_freq_list_duration*, and *nifgen.Session.freq_list_duration_quantum*.

Return type

tuple (maximum_number_of_freq_lists, minimum_frequency_list_length, maximum_frequency_list_length, minimum_frequency_list_duration, maximum_frequency_list_duration, frequency_list_duration_quantum)

WHERE

maximum_number_of_freq_lists (int):

Returns the maximum number of frequency lists that the signal generator allows. NI-FGEN obtains this value from the *nifgen.Session.max_num_freq_lists* property.

minimum_frequency_list_length (int):

Returns the minimum number of steps that the signal generator allows in a frequency list. NI-FGEN obtains this value from the *nifgen.Session.min_freq_list_length* property.

maximum_frequency_list_length (int):

Returns the maximum number of steps that the signal generator allows in a frequency list. NI-FGEN obtains this value from the *nifgen.Session.max_freq_list_length* property.

minimum_frequency_list_duration (float):

Returns the minimum duration that the signal generator allows in a step of a frequency list. NI-FGEN obtains this value from the *nifgen.Session.min_freq_list_duration* property.

maximum_frequency_list_duration (float):

Returns the maximum duration that the signal generator allows in a step of a frequency list. NI-FGEN obtains this value from the `nifgen.Session.max_freq_list_duration` property.

frequency_list_duration_quantum (float):

Returns the quantum of which all durations must be a multiple in a frequency list. NI-FGEN obtains this value from the `nifgen.Session.freq_list_duration_quantum` property.

read_current_temperature

`nifgen.Session.`**`read_current_temperature`**`()`

Reads the current onboard temperature of the device. The temperature is returned in degrees Celsius.

Return type float

Returns Returns the current temperature read from onboard temperature sensors, in degrees Celsius.

reset

`nifgen.Session.`**`reset`**`()`

Resets the instrument to a known state. This method aborts the generation, clears all routes, and resets session properties to the default values. This method does not, however, commit the session properties or configure the device hardware to its default state.

Note: For the NI 5401/5404/5411/5431, this method exhibits the same behavior as the `nifgen.Session.reset_device()` method.

reset_device

`nifgen.Session.`**`reset_device`**`()`

Performs a hard reset on the device. Generation is stopped, all routes are released, external bidirectional terminals are tristated, FPGAs are reset, hardware is configured to its default state, and all session properties are reset to their default states.

reset_with_defaults

`nifgen.Session.`**`reset_with_defaults`**`()`

Resets the instrument and reapplies initial user–specified settings from the logical name that was used to initialize the session. If the session was created without a logical name, this method is equivalent to the `nifgen.Session.reset()` method.

self_cal

`nifgen.Session.`**`self_cal`**`()`

Performs a full internal self-calibration on the device. If the calibration is successful, new calibration data and constants are stored in the onboard EEPROM.

self_test

nifgen.Session.**self_test**()

Runs the instrument self-test routine and returns the test result(s).

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

Note: When used on some signal generators, the device is reset after the `nifgen.Session.self_test()` method runs. If you use the `nifgen.Session.self_test()` method, your device may not be in its previously configured state after the method runs.

send_software_edge_trigger

nifgen.Session.**send_software_edge_trigger**(*trigger*, *trigger_id*)

Sends a command to trigger the signal generator. This VI can act as an override for an external edge trigger.

Note: This VI does not override external digital edge triggers of the NI 5401/5411/5431.

Parameters

- **trigger** (`nifgen.Trigger`) – Trigger specifies the type of software trigger to send

Defined Values
START
SCRIPT

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.

- **trigger_id** (`str`) – Trigger ID specifies the Script Trigger to use for triggering.

set_next_write_position

nifgen.Session.**set_next_write_position**(*waveform_name_or_handle*, *relative_to*, *offset*)

Sets the position in the waveform at which the next waveform data is written. This method allows you to write to arbitrary locations within the waveform. These settings apply only to the next write to the waveform specified by the waveformHandle parameter. Subsequent writes to that waveform begin where the last write left off, unless this method is called again. The waveformHandle passed in must have been created by a call to the `nifgen.Session.allocate_waveform()` method or one of the following `nifgen.Session.create_waveform()` method.

Tip: This method can be called on specific channels within your `nifgen.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[ ... ].set_next_write_position()`

To call the method on all channels, you can call it directly on the `nifgen.Session`.

Example: `my_session.set_next_write_position()`

Parameters

- **`waveform_name_or_handle`** (*`str or int`*) – The name (str) or handle (int) of an arbitrary waveform previously allocated with `nifgen.Session.allocate_named_waveform()`, `nifgen.Session.allocate_waveform()` or `nifgen.Session.create_waveform()`.
- **`relative_to`** (*`nifgen.RelativeTo`*) – Specifies the reference position in the waveform. This position and **offset** together determine where to start loading data into the waveform.

  ****Defined Values****

START (0)	Use the start of the waveform as the reference position.
CURRENT (1)	Use the current position within the waveform as the reference position.

- **`offset`** (*`int`*) – Specifies the offset from **relativeTo** at which to start loading the data into the waveform.

unlock

`nifgen.Session.`**`unlock`**`()`

Releases a lock that you acquired on an device session using `nifgen.Session.lock()`. Refer to `nifgen.Session.unlock()` for additional information on session locks.

wait_until_done

`nifgen.Session.`**`wait_until_done`**`(`*max_time=hightime.timedelta(seconds=10.0)*`)`

Waits until the device is done generating or until the maximum time has expired.

Parameters **`max_time`** (*`hightime.timedelta, datetime.timedelta, or int in milliseconds`*) – Specifies the timeout value in milliseconds.

write_script

`nifgen.Session.`**`write_script`**`(`*script*`)`

Writes a string containing one or more scripts that govern the generation of waveforms.

Tip: This method can be called on specific channels within your `nifgen.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[ ... ].write_script()`

To call the method on all channels, you can call it directly on the `nifgen.Session`.

Example: `my_session.write_script()`

Parameters **`script`** (*`str`*) – Contains the text of the script you want to use for your generation operation. Refer to scripting Instructions for more information about writing scripts.

write_waveform

`nifgen.Session.`**`write_waveform`**(*waveform_name_or_handle*, *data*)

Writes data to the waveform in onboard memory.

By default, subsequent calls to this method continue writing data from the position of the last sample written. You can set the write position and offset by calling the `nifgen.Session.set_next_write_position()` `nifgen.Session.set_next_write_position()` method.

Tip: This method can be called on specific channels within your `nifgen.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[ ... ].write_waveform()`

To call the method on all channels, you can call it directly on the `nifgen.Session`.

Example: `my_session.write_waveform()`

Parameters

- **`waveform_name_or_handle`** (*`str or int`*) – The name (str) or handle (int) of an arbitrary waveform previously allocated with `nifgen.Session.allocate_named_waveform()`, `nifgen.Session.allocate_waveform()` or `nifgen.Session.create_waveform()`.
- **`data`** (*`list of float`*) – Array of data to load into the waveform. This may be an iterable of float, or for best performance a numpy.ndarray of dtype int16 or float64.

Properties

absolute_delay

`nifgen.Session.`**`absolute_delay`**

Specifies the sub-Sample Clock delay, in seconds, to apply to the waveform. Use this property to reduce the trigger jitter when synchronizing multiple devices with NI-TClk. This property can also help maintain synchronization repeatability by writing the absolute delay value of a previous measurement to the current session. To set this property, the waveform generator must be in the Idle (Configuration) state. **Units**: seconds (s) **Valid Values**: Plus or minus half of one Sample Clock period **Default Value**: 0.0 **Supported Waveform Generators**: PXIe-5413/5423/5433

Note: If this property is set, NI-TClk cannot perform any sub-Sample Clock adjustment.

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: **Output:Absolute Delay**
- C Attribute: **NIFGEN_ATTR_ABSOLUTE_DELAY**

all_marker_events_latched_status

`nifgen.Session.`**`all_marker_events_latched_status`**

Returns a bit field of the latched status of all Marker Events. Write 0 to this property to clear the latched status of all Marker Events.

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: **Events:Marker:Advanced:All Marker Events Latched Status**
- C Attribute: **NIFGEN_ATTR_ALL_MARKER_EVENTS_LATCHED_STATUS**

all_marker_events_live_status

`nifgen.Session.`**`all_marker_events_live_status`**

Returns a bit field of the live status of all Marker Events.

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: **Events:Marker:Advanced:All Marker Events Live Status**
- C Attribute: **NIFGEN_ATTR_ALL_MARKER_EVENTS_LIVE_STATUS**

analog_data_mask

`nifgen.Session.`**`analog_data_mask`**

Specifies the mask to apply to the analog output. The masked data is replaced with the data in *`nifgen.Session.analog_static_value`*.

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: **Output:Data Mask:Analog Data Mask**
- C Attribute: **NIFGEN_ATTR_ANALOG_DATA_MASK**

analog_filter_enabled

`nifgen.Session.`**`analog_filter_enabled`**

Controls whether the signal generator applies to an analog filter to the output signal. This property is valid in arbitrary waveform, arbitrary sequence, and script modes. This property can also be used in standard method and frequency list modes for user-defined waveforms.

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: **Output:Filters:Analog Filter Enabled**
- C Attribute: **NIFGEN_ATTR_ANALOG_FILTER_ENABLED**

analog_path

`nifgen.Session.`**`analog_path`**

Specifies the analog signal path that should be used. The main path allows you to configure gain, offset, analog filter status, output impedance, and output enable. The main path has two amplifier options, high- and low-gain. The direct path presents a much smaller gain range, and you cannot adjust offset or the filter status. The direct path also provides a smaller output range but also lower distortion. NI-FGEN normally chooses the amplifier based on the user-specified gain.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.AnalogPath
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Output:Analog Path**
- C Attribute: **NIFGEN_ATTR_ANALOG_PATH**

analog_static_value

`nifgen.Session.`**`analog_static_value`**

Specifies the static value that replaces data masked by *`nifgen.Session.analog_data_mask`*.

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: **Output:Data Mask:Analog Static Value**
- C Attribute: **NIFGEN_ATTR_ANALOG_STATIC_VALUE**

arb_gain

`nifgen.Session.`**`arb_gain`**

Specifies the factor by which the signal generator scales the arbitrary waveform data. When you create arbitrary waveforms, you must first normalize the data points to the range -1.0 to +1.0. Use this property to scale the arbitrary waveform to other ranges. For example, when you set this property to 2.0, the output signal ranges from -2.0 V to +2.0 V. Use this property when *`nifgen.Session.output_mode`* is set to *`ARB`* or *`SEQ`*.

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: **Arbitrary Waveform:Gain**
- C Attribute: **NIFGEN_ATTR_ARB_GAIN**

arb_marker_position

`nifgen.Session.`**`arb_marker_position`**

Specifies the position for a marker to be asserted in the arbitrary waveform. This property defaults to -1 when no marker position is specified. Use this property when *`nifgen.Session.output_mode`* is set to *`ARB`*. Use `nifgen.Session.ExportSignal()` to export the marker signal.

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	int
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Arbitrary Waveform:Arbitrary Waveform Mode:Marker Position**
- C Attribute: **NIFGEN_ATTR_ARB_MARKER_POSITION**

arb_offset

`nifgen.Session.`**`arb_offset`**

Specifies the value that the signal generator adds to the arbitrary waveform data. When you create arbitrary waveforms, you must first normalize the data points to the range -1.0 to +1.0. Use this property to shift the arbitrary waveform range. For example, when you set this property to 1.0, the output signal ranges from 2.0 V to 0.0 V. Use this property when *`nifgen.Session.output_mode`* is set to *`ARB`* or *`SEQ`*. Units: Volts

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: **Arbitrary Waveform:Offset**
- C Attribute: **NIFGEN_ATTR_ARB_OFFSET**

arb_repeat_count

`nifgen.Session.`**`arb_repeat_count`**

Specifies number of times to repeat the arbitrary waveform when the triggerMode parameter of `nifgen.Session.ConfigureTriggerMode()` is set to *`SINGLE`* or *`STEPPED`*. This property is ignored if the triggerMode parameter is set to *`CONTINUOUS`* or *`BURST`*. Use this property when *`nifgen.Session.output_mode`* is set to *`ARB`*. When used during streaming, this property specifies the number of times to repeat the streaming waveform (the onboard memory allocated for streaming). For more information about streaming, refer to the Streaming topic.

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: **Arbitrary Waveform:Arbitrary Waveform Mode:Repeat Count**
- C Attribute: **NIFGEN_ATTR_ARB_REPEAT_COUNT**

arb_sample_rate

`nifgen.Session.`**`arb_sample_rate`**

Specifies the rate at which the signal generator outputs the points in arbitrary waveforms. Use this property when *`nifgen.Session.output_mode`* is set to *`ARB`* or *`SEQ`*. Units: Samples/s

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: **Clocks:Sample Clock:Rate**
- C Attribute: **NIFGEN_ATTR_ARB_SAMPLE_RATE**

arb_sequence_handle

`nifgen.Session.`**`arb_sequence_handle`**

This channel-based property identifies which sequence the signal generator produces. You can create multiple sequences using *`nifgen.Session.create_arb_sequence()`*. *`nifgen.Session.create_arb_sequence()`* returns a handle that you can use to identify the particular sequence. To configure the signal generator to produce a particular sequence, set this property to the sequence handle. Use this property only when *`nifgen.Session.output_mode`* is set to *`SEQ`*.

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: **Arbitrary Waveform:Arbitrary Sequence Mode:Arbitrary Sequence Handle**
- C Attribute: **NIFGEN_ATTR_ARB_SEQUENCE_HANDLE**

arb_waveform_handle

`nifgen.Session.`**`arb_waveform_handle`**

Selects which arbitrary waveform the signal generator produces. You can create multiple arbitrary waveforms using one of the following niFgen Create Waveform methods: `nifgen.Session.create_waveform()` `nifgen.Session.create_waveform()` *`nifgen.Session.create_waveform_from_file_i16()`* *`nifgen.Session.create_waveform_from_file_f64()`* `nifgen.Session.CreateWaveformFromFileHWS()` These methods return a handle that you can use to identify the particular waveform. To configure the signal generator to produce a particular waveform, set this property to the waveform handle. Use this property only when *`nifgen.Session.output_mode`* is set to *`ARB`*.

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	int
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Arbitrary Waveform:Arbitrary Waveform Mode:Arbitrary Waveform Handle**
- C Attribute: **NIFGEN_ATTR_ARB_WAVEFORM_HANDLE**

aux_power_enabled

nifgen.Session.**aux_power_enabled**

Controls the specified auxiliary power pin. Setting this property to TRUE energizes the auxiliary power when the session is committed. When this property is FALSE, the power pin of the connector outputs no power.

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: **Output:Advanced:AUX Power Enabled**
- C Attribute: **NIFGEN_ATTR_AUX_POWER_ENABLED**

bus_type

nifgen.Session.**bus_type**

The bus type of the signal generator.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.BusType
Permissions	read only
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Instrument:Bus Type**
- C Attribute: **NIFGEN_ATTR_BUS_TYPE**

channel_delay

`nifgen.Session.`**`channel_delay`**

Specifies, in seconds, the delay to apply to the analog output of the channel specified by the channel string. You can use the channel delay to configure the timing relationship between channels on a multichannel device. Values for this property can be zero or positive. A value of zero indicates that the channels are aligned. A positive value delays the analog output by the specified number of seconds.

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: **Output:Channel Delay**
- C Attribute: **NIFGEN_ATTR_CHANNEL_DELAY**

clock_mode

`nifgen.Session.`**`clock_mode`**

Controls which clock mode is used for the signal generator. For signal generators that support it, this property allows switching the sample clock to High-Resolution mode. When in Divide-Down mode, the sample rate can only be set to certain frequences, based on dividing down the update clock. However, in High-Resolution mode, the sample rate may be set to any value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ClockMode
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocks:Sample Clock:Mode**
- C Attribute: **NIFGEN_ATTR_CLOCK_MODE**

common_mode_offset

`nifgen.Session.`**`common_mode_offset`**

Specifies, in volts, the value the signal generator adds to or subtracts from the arbitrary waveform data. This property applies only when you set the *`nifgen.Session.terminal_configuration`* property to *`DIFFERENTIAL`*. Common mode offset is applied to the signals generated at each differential output terminal.

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: **Output:Common Mode Offset**
- C Attribute: **NIFGEN_ATTR_COMMON_MODE_OFFSET**

data_marker_events_count

`nifgen.Session.`**`data_marker_events_count`**

Returns the number of Data Marker Events supported by the device.

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: **Instrument:Data Marker Events Count**
- C Attribute: **NIFGEN_ATTR_DATA_MARKER_EVENTS_COUNT**

data_marker_event_data_bit_number

`nifgen.Session.`**`data_marker_event_data_bit_number`**

Specifies the bit number to assign to the Data Marker Event.

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: **Events:Data Marker:Data Bit Number**
- C Attribute: **NIFGEN_ATTR_DATA_MARKER_EVENT_DATA_BIT_NUMBER**

data_marker_event_level_polarity

nifgen.Session.**data_marker_event_level_polarity**

Specifies the output polarity of the Data marker event.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.DataMarkerEventLevelPolarity
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Events:Data Marker:Level:Active Level**
- C Attribute: **NIFGEN_ATTR_DATA_MARKER_EVENT_LEVEL_POLARITY**

data_marker_event_output_terminal

nifgen.Session.**data_marker_event_output_terminal**

Specifies the destination terminal for the Data Marker 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: **Events:Data Marker:Output Terminal**
- C Attribute: **NIFGEN_ATTR_DATA_MARKER_EVENT_OUTPUT_TERMINAL**

data_transfer_block_size

nifgen.Session.**data_transfer_block_size**

The number of samples at a time to download to onboard memory. Useful when the total data to be transferred to onboard memory is large.

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: **Arbitrary Waveform:Data Transfer:Data Transfer Block Size**
- C Attribute: **NIFGEN_ATTR_DATA_TRANSFER_BLOCK_SIZE**

data_transfer_maximum_bandwidth

`nifgen.Session.`**`data_transfer_maximum_bandwidth`**

Specifies the maximum amount of bus bandwidth (in bytes per second) to use for data transfers. The signal generator limits data transfer speeds on the PCIe bus to the value you specify for this property. Set this property to optimize bus bandwidth usage for multi-device streaming applications by preventing the signal generator from consuming all of the available bandwidth on a PCI express link when waveforms are being written to the onboard memory of the device.

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: **Arbitrary Waveform:Data Transfer:Maximum Bandwidth**
- C Attribute: **NIFGEN_ATTR_DATA_TRANSFER_MAXIMUM_BANDWIDTH**

data_transfer_maximum_in_flight_reads

`nifgen.Session.`**`data_transfer_maximum_in_flight_reads`**

Specifies the maximum number of concurrent PCI Express read requests the signal generator can issue. When transferring data from computer memory to device onboard memory across the PCI Express bus, the signal generator can issue multiple memory reads at the same time. In general, the larger the number of read requests, the more efficiently the device uses the bus because the multiple read requests keep the data flowing, even in a PCI Express topology that has high latency due to PCI Express switches in the data path. Most NI devices can issue a large number of read requests (typically 8 or 16). By default, this property is set to the highest value the signal generator supports. If other devices in your system cannot tolerate long data latencies, it may be helpful to decrease the number of in-flight read requests the NI signal generator issues. This helps to reduce the amount of data the signal generator reads at one time.

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: **Arbitrary Waveform:Data Transfer:Advanced:Maximum In-Flight Read Requests**
- C Attribute: **NIFGEN_ATTR_DATA_TRANSFER_MAXIMUM_IN_FLIGHT_READS**

data_transfer_preferred_packet_size

`nifgen.Session.`**`data_transfer_preferred_packet_size`**

Specifies the preferred size of the data field in a PCI Express read request packet. In general, the larger the packet size, the more efficiently the device uses the bus. By default, NI signal generators use the largest packet size allowed by the system. However, due to different system implementations, some systems may perform better with smaller packet sizes. Recommended values for this property are powers of two between 64 and 512. In some cases, the signal generator generates packets smaller than the preferred size you set with this property. You cannot change this property while the device is generating a waveform. If you want to change the device configuration, call the *`nifgen.Session.abort()`* method or wait for the generation to complete.

Note: :

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: **Arbitrary Waveform:Data Transfer:Advanced:Preferred Packet Size**
- C Attribute: **NIFGEN_ATTR_DATA_TRANSFER_PREFERRED_PACKET_SIZE**

digital_data_mask

`nifgen.Session.`**`digital_data_mask`**

Specifies the mask to apply to the output on the digital connector. The masked data is replaced with the data in *`nifgen.Session.digital_static_value`*.

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: **Output:Data Mask:Digital Data Mask**
- C Attribute: **NIFGEN_ATTR_DIGITAL_DATA_MASK**

digital_edge_script_trigger_edge

nifgen.Session.**digital_edge_script_trigger_edge**

Specifies the active edge for the Script trigger. This property is used when *nifgen.Session.script_trigger_type* is set to Digital Edge.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ScriptTriggerDigitalEdgeEdge
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggers:Script:Digital Edge:Edge**
- C Attribute: **NIFGEN_ATTR_DIGITAL_EDGE_SCRIPT_TRIGGER_EDGE**

digital_edge_script_trigger_source

nifgen.Session.**digital_edge_script_trigger_source**

Specifies the source terminal for the Script trigger. This property is used when *nifgen.Session.script_trigger_type* is set to Digital Edge.

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: **Triggers:Script:Digital Edge:Source**
- C Attribute: **NIFGEN_ATTR_DIGITAL_EDGE_SCRIPT_TRIGGER_SOURCE**

digital_edge_start_trigger_edge

`nifgen.Session.`**`digital_edge_start_trigger_edge`**

Specifies the active edge for the Start trigger. This property is used only when *`nifgen.Session.start_trigger_type`* is set to Digital Edge.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.StartTriggerDigitalEdgeEdge
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggers:Start:Digital Edge:Edge**
- C Attribute: **NIFGEN_ATTR_DIGITAL_EDGE_START_TRIGGER_EDGE**

digital_edge_start_trigger_source

`nifgen.Session.`**`digital_edge_start_trigger_source`**

Specifies the source terminal for the Start trigger. This property is used only when *`nifgen.Session.start_trigger_type`* is set to Digital Edge.

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: **Triggers:Start:Digital Edge:Source**
- C Attribute: **NIFGEN_ATTR_DIGITAL_EDGE_START_TRIGGER_SOURCE**

digital_filter_enabled

`nifgen.Session.`**`digital_filter_enabled`**

Controls whether the signal generator applies a digital filter to the output signal. This property is valid in arbitrary waveform, arbitrary sequence, and script modes. This property can also be used in standard method and frequency list modes for user-defined waveforms.

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: **Output:Filters:Digital Filter Enabled**
- C Attribute: **NIFGEN_ATTR_DIGITAL_FILTER_ENABLED**

digital_filter_interpolation_factor

`nifgen.Session.`**`digital_filter_interpolation_factor`**

This property only affects the device when *`nifgen.Session.digital_filter_enabled`* is set to True. If you do not set this property directly, NI-FGEN automatically selects the maximum interpolation factor allowed for the current sample rate. Valid values are 2, 4, and 8.

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: **Output:Filters:Digital Filter Interpolation Factor**
- C Attribute: **NIFGEN_ATTR_DIGITAL_FILTER_INTERPOLATION_FACTOR**

digital_gain

`nifgen.Session.`**`digital_gain`**

Specifies a factor by which the signal generator digitally multiplies generated data before converting it to an analog signal in the DAC. For a digital gain greater than 1.0, the product of digital gain times the generated data must be inside the range plus or minus 1.0 (assuming floating point data). If the product exceeds these limits, the signal generator clips the output signal, and an error results. Some signal generators support both digital gain and an analog gain (analog gain is specified with the *`nifgen.Session.func_amplitude`* property or the *`nifgen.Session.arb_gain`* property). Digital gain can be changed during generation without the glitches that may occur when changing analog gains, due to relay switching. However, the DAC output resolution is a method of analog gain, so only analog gain makes full use of the resolution of the DAC.

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: **Output:Digital Gain**
- C Attribute: **NIFGEN_ATTR_DIGITAL_GAIN**

digital_pattern_enabled

`nifgen.Session.`**`digital_pattern_enabled`**

Controls whether the signal generator generates a digital pattern of the output signal.

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: **Output:Advanced:Digital Pattern Enabled**
- C Attribute: **NIFGEN_ATTR_DIGITAL_PATTERN_ENABLED**

digital_static_value

`nifgen.Session.`**`digital_static_value`**

Specifies the static value that replaces data masked by *`nifgen.Session.digital_data_mask`*.

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: **Output:Data Mask:Digital Static Value**
- C Attribute: **NIFGEN_ATTR_DIGITAL_STATIC_VALUE**

done_event_output_terminal

`nifgen.Session.done_event_output_terminal`

Specifies the destination terminal for the Done 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: **Events:Done:Output Terminal**
- C Attribute: **NIFGEN_ATTR_DONE_EVENT_OUTPUT_TERMINAL**

driver_setup

`nifgen.Session.driver_setup`

Specifies the driver setup portion of the option string that was passed into the `nifgen.Session.InitWithOptions()` method.

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:

- C Attribute: **NIFGEN_ATTR_DRIVER_SETUP**

exported_onboard_reference_clock_output_terminal

`nifgen.Session.exported_onboard_reference_clock_output_terminal`

Specifies the terminal to which to export the Onboard Reference 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: **Clocks:Reference Clock:Onboard Reference Clock:Export Output Terminal**
- C Attribute: **NIFGEN_ATTR_EXPORTED_ONBOARD_REFERENCE_CLOCK_OUTPUT_TERMINAL**

exported_reference_clock_output_terminal

`nifgen.Session.`**`exported_reference_clock_output_terminal`**

Specifies the terminal to which to export the Reference 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: **Clocks:Reference Clock:Export Output Terminal**
- C Attribute: **NIFGEN_ATTR_EXPORTED_REFERENCE_CLOCK_OUTPUT_TERMINAL**

exported_sample_clock_divisor

`nifgen.Session.`**`exported_sample_clock_divisor`**

Specifies the factor by which to divide the Sample clock, also known as the Update clock, before it is exported. To export the Sample clock, use the `nifgen.Session.ExportSignal()` method or the *`nifgen.Session.exported_sample_clock_output_terminal`* property.

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	int
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocks:Sample Clock:Exported Sample Clock Divisor**
- C Attribute: **NIFGEN_ATTR_EXPORTED_SAMPLE_CLOCK_DIVISOR**

exported_sample_clock_output_terminal

nifgen.Session.**exported_sample_clock_output_terminal**

Specifies the terminal to which to export the Sample 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: **Clocks:Sample Clock:Export Output Terminal**
- C Attribute: **NIFGEN_ATTR_EXPORTED_SAMPLE_CLOCK_OUTPUT_TERMINAL**

exported_sample_clock_timebase_divisor

nifgen.Session.**exported_sample_clock_timebase_divisor**

Specifies the factor by which to divide the sample clock timebase (board clock) before it is exported. To export the Sample clock timebase, use the `nifgen.Session.ExportSignal()` method or the *nifgen.Session.exported_sample_clock_timebase_output_terminal* property.

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	int
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocks:Sample Clock Timebase:Exported Sample Clock Timebase Divisor**
- C Attribute: **NIFGEN_ATTR_EXPORTED_SAMPLE_CLOCK_TIMEBASE_DIVISOR**

exported_sample_clock_timebase_output_terminal

nifgen.Session.**exported_sample_clock_timebase_output_terminal**

Specifies the terminal to which to export the Sample clock timebase. If you specify a divisor with the *nifgen.Session.exported_sample_clock_timebase_divisor* property, the Sample clock exported with the *nifgen.Session.exported_sample_clock_timebase_output_terminal* property is the value of the Sample clock timebase after it is divided-down. For a list of the terminals available on your device, refer to the Device Routes tab in MAX. To change the device configuration, call *nifgen.Session.abort()* or wait for the generation to complete.

Note: The signal generator must not be in the Generating state when you change this property.

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: **Clocks:Sample Clock Timebase:Export Output Terminal**
- C Attribute: **NIFGEN_ATTR_EXPORTED_SAMPLE_CLOCK_TIMEBASE_OUTPUT_TERMINAL**

exported_script_trigger_output_terminal

`nifgen.Session.`**`exported_script_trigger_output_terminal`**

Specifies the output terminal for the exported Script trigger. Setting this property to an empty string means that when you commit the session, the signal is removed from that terminal and, if possible, the terminal is tristated.

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: **Triggers:Script:Output Terminal**
- C Attribute: **NIFGEN_ATTR_EXPORTED_SCRIPT_TRIGGER_OUTPUT_TERMINAL**

exported_start_trigger_output_terminal

`nifgen.Session.`**`exported_start_trigger_output_terminal`**

Specifies the destination terminal for exporting the Start trigger.

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: **Triggers:Start:Output Terminal**
- C Attribute: **NIFGEN_ATTR_EXPORTED_START_TRIGGER_OUTPUT_TERMINAL**

external_clock_delay_binary_value

`nifgen.Session.`**`external_clock_delay_binary_value`**

Binary value of the external clock delay.

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: **Clocks:Advanced:External Clock Delay Binary Value**
- C Attribute: **NIFGEN_ATTR_EXTERNAL_CLOCK_DELAY_BINARY_VALUE**

external_sample_clock_multiplier

`nifgen.Session.`**`external_sample_clock_multiplier`**

Specifies a multiplication factor to use to obtain a desired sample rate from an external Sample clock. The resulting sample rate is equal to this factor multiplied by the external Sample clock rate. You can use this property to generate samples at a rate higher than your external clock rate. When using this property, you do not need to explicitly set the external clock rate.

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: **Clocks:Advanced:External Sample Clock Multiplier**
- C Attribute: **NIFGEN_ATTR_EXTERNAL_SAMPLE_CLOCK_MULTIPLIER**

file_transfer_block_size

`nifgen.Session.`**`file_transfer_block_size`**

The number of samples at a time to read from the file and download to onboard memory. Used in conjunction with the Create From File and Write From File methods.

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: **Arbitrary Waveform:Data Transfer:File Transfer Block Size**
- C Attribute: **NIFGEN_ATTR_FILE_TRANSFER_BLOCK_SIZE**

filter_correction_frequency

`nifgen.Session.`**`filter_correction_frequency`**

Controls the filter correction frequency of the analog filter. This property corrects for the ripples in the analog filter frequency response at the frequency specified. For standard waveform output, the filter correction frequency should be set to be the same as the frequency of the standard waveform. To have no filter correction, set this property to 0 Hz.

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: **Instrument:5401/5411/5431:Filter Correction Frequency**
- C Attribute: **NIFGEN_ATTR_FILTER_CORRECTION_FREQUENCY**

flatness_correction_enabled

`nifgen.Session.`**`flatness_correction_enabled`**

When True, the signal generator applies a flatness correction factor to the generated sine wave in order to ensure the same output power level at all frequencies. This property should be set to False when performing Flatness Calibration.

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: **Output:Filters:Flatness Correction Enabled**
- C Attribute: **NIFGEN_ATTR_FLATNESS_CORRECTION_ENABLED**

fpga_bitfile_path

`nifgen.Session.`**`fpga_bitfile_path`**

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: **Instrument:FPGA Bitfile Path**
- C Attribute: **NIFGEN_ATTR_FPGA_BITFILE_PATH**

freq_list_duration_quantum

`nifgen.Session.`**`freq_list_duration_quantum`**

Returns the quantum of which all durations must be a multiple in a frequency list.

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: **Standard Function:Frequency List Mode:Frequency List Duration Quantum**
- C Attribute: **NIFGEN_ATTR_FREQ_LIST_DURATION_QUANTUM**

freq_list_handle

nifgen.Session.**freq_list_handle**

Sets which frequency list the signal generator produces. Create a frequency list using *nifgen.Session.create_freq_list()*. *nifgen.Session.create_freq_list()* returns a handle that you can use to identify the list.

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: **Standard Function:Frequency List Mode:Frequency List Handle**
- C Attribute: **NIFGEN_ATTR_FREQ_LIST_HANDLE**

func_amplitude

nifgen.Session.**func_amplitude**

Controls the amplitude of the standard waveform that the signal generator produces. This value is the amplitude at the output terminal. For example, to produce a waveform ranging from -5.00 V to +5.00 V, set the amplitude to 10.00 V. set the Waveform parameter to *DC*. Units: Vpk-pk

Note: This parameter does not affect signal generator behavior when you

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: **Standard Function:Amplitude**
- C Attribute: **NIFGEN_ATTR_FUNC_AMPLITUDE**

func_buffer_size

nifgen.Session.**func_buffer_size**

This property contains the number of samples used in the standard method waveform buffer. This property is only valid on devices that implement standard method mode in software, and is read-only for all other devices. implementation of Standard Method Mode on your device.

Note: Refer to the Standard Method Mode topic for more information on the

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: **Standard Function:Standard Function Mode:Buffer Size**
- C Attribute: **NIFGEN_ATTR_FUNC_BUFFER_SIZE**

func_dc_offset

`nifgen.Session.`**`func_dc_offset`**

Controls the DC offset of the standard waveform that the signal generator produces. This value is the offset at the output terminal. The value is the offset from ground to the center of the waveform that you specify with the Waveform parameter. For example, to configure a waveform with an amplitude of 10.00 V to range from 0.00 V to +10.00 V, set DC Offset to 5.00 V. Units: volts

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: **Standard Function:DC Offset**
- C Attribute: **NIFGEN_ATTR_FUNC_DC_OFFSET**

func_duty_cycle_high

`nifgen.Session.`**`func_duty_cycle_high`**

Controls the duty cycle of the square wave the signal generator produces. Specify this property as a percentage of the time the square wave is high in a cycle. set the Waveform parameter to *SQUARE*. Units: Percentage of time the waveform is high

Note: This parameter only affects signal generator behavior when you

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: **Standard Function:Duty Cycle High**
- C Attribute: **NIFGEN_ATTR_FUNC_DUTY_CYCLE_HIGH**

func_frequency

`nifgen.Session.`**`func_frequency`**

Controls the frequency of the standard waveform that the signal generator produces. Units: hertz (1) This parameter does not affect signal generator behavior when you set the Waveform parameter of the *`nifgen.Session.configure_standard_waveform()`* method to *`DC`*. (2) For *`SINE`*, the range is between 0 MHz and 16 MHz, but the range is between 0 MHz and 1 MHz for all other waveforms.

Note: :

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: **Standard Function:Standard Function Mode:Frequency**
- C Attribute: **NIFGEN_ATTR_FUNC_FREQUENCY**

func_max_buffer_size

`nifgen.Session.`**`func_max_buffer_size`**

This property sets the maximum number of samples that can be used in the standard method waveform buffer. Increasing this value may increase the quality of the waveform. This property is only valid on devices that implement standard method mode in software, and is read-only for all other devices. implementation of Standard Method Mode on your device.

Note: Refer to the Standard Method Mode topic for more information on the

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: **Standard Function:Standard Function Mode:Maximum Buffer Size**
- C Attribute: **NIFGEN_ATTR_FUNC_MAX_BUFFER_SIZE**

func_start_phase

`nifgen.Session.`**`func_start_phase`**

Controls horizontal offset of the standard waveform the signal generator produces. Specify this property in degrees of one waveform cycle. A start phase of 180 degrees means output generation begins halfway through the waveform. A start phase of 360 degrees offsets the output by an entire waveform cycle, which is identical to a start phase of 0 degrees. set the Waveform parameter to *`DC`*. Units: Degrees of one cycle

Note: This parameter does not affect signal generator behavior when you

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: **Standard Function:Start Phase**
- C Attribute: **NIFGEN_ATTR_FUNC_START_PHASE**

func_waveform

`nifgen.Session.`**`func_waveform`**

This channel-based property specifies which standard waveform the signal generator produces. Use this property only when *`nifgen.Session.output_mode`* is set to *`FUNC`*. *`SINE`* - Sinusoid waveform *`SQUARE`* - Square waveform *`TRIANGLE`* - Triangle waveform *`RAMP_UP`* - Positive ramp waveform *`RAMP_DOWN`* - Negative ramp waveform *`DC`* - Constant voltage *`NOISE`* - White noise *`USER`* - User-defined waveform as defined with *`nifgen.Session.define_user_standard_waveform()`*

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.Waveform
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Standard Function:Waveform**
- C Attribute: **NIFGEN_ATTR_FUNC_WAVEFORM**

idle_behavior

`nifgen.Session.`**`idle_behavior`**

Specifies the behavior of the output during the Idle state. The output can be configured to hold the last generated voltage before entering the Idle state or jump to the Idle Value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.IdleBehavior
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Output:Advanced:Idle Behavior**
- C Attribute: **NIFGEN_ATTR_IDLE_BEHAVIOR**

idle_value

`nifgen.Session.`**`idle_value`**

Specifies the value to generate in the Idle state. The Idle Behavior must be configured to jump to this value.

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: **Output:Advanced:Idle Value**
- C Attribute: **NIFGEN_ATTR_IDLE_VALUE**

instrument_firmware_revision

nifgen.Session.**instrument_firmware_revision**

A string that contains the firmware revision information for the device that you are currently using.

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: **Instrument:Inherent IVI Attributes:Instrument Identification:Firmware Revision**
- C Attribute: **NIFGEN_ATTR_INSTRUMENT_FIRMWARE_REVISION**

instrument_manufacturer

nifgen.Session.**instrument_manufacturer**

A string that contains the name of the device manufacturer you are currently using.

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: **Instrument:Inherent IVI Attributes:Instrument Identification:Manufacturer**
- C Attribute: **NIFGEN_ATTR_INSTRUMENT_MANUFACTURER**

instrument_model

nifgen.Session.**instrument_model**

A string that contains the model number or name of the device that you are currently using.

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: **Instrument:Inherent IVI Attributes:Instrument Identification:Model**
- C Attribute: **NIFGEN_ATTR_INSTRUMENT_MODEL**

io_resource_descriptor

`nifgen.Session.`**`io_resource_descriptor`**

Indicates the resource descriptor that NI-FGEN uses to identify the physical device. If you initialize NI-FGEN with a logical name, this property contains the resource descriptor that corresponds to the entry in the IVI Configuration Utility. If you initialize NI-FGEN with the resource descriptor, this property contains that value.

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: **Instrument:Inherent IVI Attributes:Advanced Session Information:Resource Descriptor**
- C Attribute: **NIFGEN_ATTR_IO_RESOURCE_DESCRIPTOR**

load_impedance

`nifgen.Session.`**`load_impedance`**

This channel-based property specifies the load impedance connected to the analog output of the channel. If you set this property to `NIFGEN_VAL_MATCHED_LOAD_IMPEDANCE` (-1.0), NI-FGEN assumes that the load impedance matches the output impedance. NI-FGEN compensates to give the desired peak-to-peak voltage amplitude or arbitrary gain (relative to 1 V).

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	float
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Output:Load Impedance**
- C Attribute: **NIFGEN_ATTR_LOAD_IMPEDANCE**

logical_name

`nifgen.Session.`**`logical_name`**

A string containing the logical name that you specified when opening the current IVI session. You may pass a logical name to `nifgen.Session.init()` or `nifgen.Session.InitWithOptions()`. 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: **Instrument:Inherent IVI Attributes:Advanced Session Information:Logical Name**
- C Attribute: **NIFGEN_ATTR_LOGICAL_NAME**

marker_events_count

`nifgen.Session.`**`marker_events_count`**

Returns the number of markers supported by the device. Use this property when *`nifgen.Session.output_mode`* is set to *`SCRIPT`*.

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: **Instrument:Marker Events Count**
- C Attribute: **NIFGEN_ATTR_MARKER_EVENTS_COUNT**

marker_event_output_terminal

`nifgen.Session.marker_event_output_terminal`

Specifies the destination terminal for the Marker 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: **Events:Marker:Output Terminal**
- C Attribute: **NIFGEN_ATTR_MARKER_EVENT_OUTPUT_TERMINAL**

max_freq_list_duration

`nifgen.Session.max_freq_list_duration`

Returns the maximum duration of any one step in the frequency list.

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: **Standard Function:Frequency List Mode:Maximum Frequency List Duration**
- C Attribute: **NIFGEN_ATTR_MAX_FREQ_LIST_DURATION**

max_freq_list_length

`nifgen.Session.max_freq_list_length`

Returns the maximum number of steps that can be in a frequency list.

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: **Standard Function:Frequency List Mode:Maximum Frequency List Length**
- C Attribute: **NIFGEN_ATTR_MAX_FREQ_LIST_LENGTH**

max_loop_count

`nifgen.Session.max_loop_count`

Returns the maximum number of times that the signal generator can repeat a waveform in a sequence. Typically, this value is constant for the signal generator.

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: **Arbitrary Waveform:Arbitrary Sequence Mode:Max Loop Count**
- C Attribute: **NIFGEN_ATTR_MAX_LOOP_COUNT**

max_num_freq_lists

`nifgen.Session.max_num_freq_lists`

Returns the maximum number of frequency lists the signal generator allows.

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: **Standard Function:Frequency List Mode:Maximum Number Of Frequency Lists**
- C Attribute: **NIFGEN_ATTR_MAX_NUM_FREQ_LISTS**

max_num_sequences

`nifgen.Session.max_num_sequences`

Returns the maximum number of arbitrary sequences that the signal generator allows. Typically, this value is constant for the signal generator.

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: **Arbitrary Waveform:Arbitrary Sequence Mode:Max Number of Sequences**
- C Attribute: **NIFGEN_ATTR_MAX_NUM_SEQUENCES**

max_num_waveforms

`nifgen.Session.max_num_waveforms`

Returns the maximum number of arbitrary waveforms that the signal generator allows. Typically, this value is constant for the signal generator.

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: **Arbitrary Waveform:Capabilities:Max Number of Waveforms**
- C Attribute: **NIFGEN_ATTR_MAX_NUM_WAVEFORMS**

max_sequence_length

`nifgen.Session.max_sequence_length`

Returns the maximum number of arbitrary waveforms that the signal generator allows in a sequence. Typically, this value is constant for the signal generator.

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: **Arbitrary Waveform:Arbitrary Sequence Mode:Max Sequence Length**
- C Attribute: **NIFGEN_ATTR_MAX_SEQUENCE_LENGTH**

max_waveform_size

`nifgen.Session.`**`max_waveform_size`**

Returns the size, in samples, of the largest waveform that can be created. This property reflects the space currently available, taking into account previously allocated waveforms and instructions.

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: **Arbitrary Waveform:Capabilities:Max Waveform Size**
- C Attribute: **NIFGEN_ATTR_MAX_WAVEFORM_SIZE**

memory_size

`nifgen.Session.`**`memory_size`**

The total amount of memory, in bytes, on the signal generator.

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: **Instrument:Memory Size**
- C Attribute: **NIFGEN_ATTR_MEMORY_SIZE**

min_freq_list_duration

`nifgen.Session.`**`min_freq_list_duration`**

Returns the minimum number of steps that can be in a frequency list.

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: **Standard Function:Frequency List Mode:Minimum Frequency List Duration**
- C Attribute: **NIFGEN_ATTR_MIN_FREQ_LIST_DURATION**

min_freq_list_length

`nifgen.Session.`**`min_freq_list_length`**

Returns the minimum number of frequency lists that the signal generator allows.

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: **Standard Function:Frequency List Mode:Minimum Frequency List Length**
- C Attribute: **NIFGEN_ATTR_MIN_FREQ_LIST_LENGTH**

min_sequence_length

`nifgen.Session.`**`min_sequence_length`**

Returns the minimum number of arbitrary waveforms that the signal generator allows in a sequence. Typically, this value is constant for the signal generator.

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: **Arbitrary Waveform:Arbitrary Sequence Mode:Min Sequence Length**
- C Attribute: **NIFGEN_ATTR_MIN_SEQUENCE_LENGTH**

min_waveform_size

`nifgen.Session.`**`min_waveform_size`**

Returns the minimum number of points that the signal generator allows in an arbitrary waveform. Typically, this value is constant for the signal generator.

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: **Arbitrary Waveform:Capabilities:Min Waveform Size**
- C Attribute: **NIFGEN_ATTR_MIN_WAVEFORM_SIZE**

module_revision

`nifgen.Session.`**`module_revision`**

A string that contains the module revision for the device that you are currently using.

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: **Instrument:Inherent IVI Attributes:Instrument Identification:Module Revision**
- C Attribute: **NIFGEN_ATTR_MODULE_REVISION**

channel_count

`nifgen.Session.channel_count`

Indicates the number of channels that the specific instrument driver supports. For each property for which IVI_VAL_MULTI_CHANNEL is set, 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: **Instrument:Inherent IVI Attributes:Driver Capabilities:Channel Count**
- C Attribute: **NIFGEN_ATTR_NUM_CHANNELS**

output_enabled

`nifgen.Session.output_enabled`

This channel-based property specifies whether the signal that the signal generator produces appears at the output connector.

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: **Output:Output Enabled**
- C Attribute: **NIFGEN_ATTR_OUTPUT_ENABLED**

output_impedance

`nifgen.Session.output_impedance`

This channel-based property specifies the signal generator output impedance at the output connector. NI signal sources modules have an output impedance of 50 ohms and an optional 75 ohms on select modules. If the load impedance matches the output impedance, then the voltage at the signal output connector is at the needed level. The voltage at the signal output connector varies with load output impedance, up to doubling the voltage for a high-impedance load.

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: **Output:Output Impedance**
- C Attribute: **NIFGEN_ATTR_OUTPUT_IMPEDANCE**

output_mode

`nifgen.Session.`**`output_mode`**

Sets which output mode the signal generator will use. The value you specify determines which methods and properties you use to configure the waveform the signal generator produces.

Note: The signal generator must not be in the Generating state when you change this property. To change the device configuration, call *`nifgen.Session.abort()`* or wait for the generation to complete.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.OutputMode
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Output:Output Mode**
- C Attribute: **NIFGEN_ATTR_OUTPUT_MODE**

ready_for_start_event_output_terminal

`nifgen.Session.`**`ready_for_start_event_output_terminal`**

Specifies the destination terminal for the Ready for Start 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: **Events:Ready For Start:Output Terminal**
- C Attribute: **NIFGEN_ATTR_READY_FOR_START_EVENT_OUTPUT_TERMINAL**

reference_clock_source

`nifgen.Session.reference_clock_source`

Specifies the reference clock source used by the signal generator. The signal generator derives the frequencies and sample rates that it uses to generate waveforms from the source you specify. For example, when you set this property to ClkIn, the signal generator uses the signal it receives at the CLK IN front panel connector as the Reference clock. To change the device configuration, call `nifgen.Session.abort()` or wait for the generation to complete.

Note: The signal generator must not be in the Generating state when you change this property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ReferenceClockSource
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocks:Reference Clock:Source**
- C Attribute: **NIFGEN_ATTR_REFERENCE_CLOCK_SOURCE**

ref_clock_frequency

`nifgen.Session.ref_clock_frequency`

Sets the frequency of the signal generator reference clock. The signal generator uses the reference clock to derive frequencies and sample rates when generating output.

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: **Clocks:Reference Clock:Frequency**
- C Attribute: **NIFGEN_ATTR_REF_CLOCK_FREQUENCY**

sample_clock_source

nifgen.Session.**sample_clock_source**

Specifies the Sample clock source. If you specify a divisor with the *nifgen.Session.exported_sample_clock_divisor* property, the Sample clock exported with the *nifgen.Session.exported_sample_clock_output_terminal* property is the value of the Sample clock after it is divided-down. For a list of the terminals available on your device, refer to the Device Routes tab in MAX. To change the device configuration, call *nifgen.Session.abort()* or wait for the generation to complete.

Note: The signal generator must not be in the Generating state when you change this property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.SampleClockSource
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocks:Sample Clock:Source**
- C Attribute: **NIFGEN_ATTR_SAMPLE_CLOCK_SOURCE**

sample_clock_timebase_rate

nifgen.Session.**sample_clock_timebase_rate**

Specifies the Sample clock timebase rate. This property applies only to external Sample clock timebases. To change the device configuration, call *nifgen.Session.abort()* or wait for the generation to complete.

Note: The signal generator must not be in the Generating state when you change this 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:

- LabVIEW Property: **Clocks:Sample Clock Timebase:Rate**
- C Attribute: **NIFGEN_ATTR_SAMPLE_CLOCK_TIMEBASE_RATE**

sample_clock_timebase_source

`nifgen.Session.`**`sample_clock_timebase_source`**

Specifies the Sample Clock Timebase source. To change the device configuration, call the *`nifgen.Session.abort()`* method or wait for the generation to complete.

Note: The signal generator must not be in the Generating state when you change this property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.SampleClockTimebaseSource
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Clocks:Sample Clock Timebase:Source**
- C Attribute: **NIFGEN_ATTR_SAMPLE_CLOCK_TIMEBASE_SOURCE**

script_to_generate

`nifgen.Session.`**`script_to_generate`**

Specifies which script the generator produces. To configure the generator to run a particular script, set this property to the name of the script. Use *`nifgen.Session.write_script()`* to create multiple scripts. Use this property when *`nifgen.Session.output_mode`* is set to *`SCRIPT`*.

Note: The signal generator must not be in the Generating state when you change this property. To change the device configuration, call *`nifgen.Session.abort()`* or wait for the generation to complete.

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: **Arbitrary Waveform:Script Mode:Script to Generate**
- C Attribute: **NIFGEN_ATTR_SCRIPT_TO_GENERATE**

script_triggers_count

`nifgen.Session.`**`script_triggers_count`**

Specifies the number of Script triggers supported by the device. Use this property when *`nifgen.Session.output_mode`* is set to *`SCRIPT`*.

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: **Instrument:Script Triggers Count**
- C Attribute: **NIFGEN_ATTR_SCRIPT_TRIGGERS_COUNT**

script_trigger_type

`nifgen.Session.`**`script_trigger_type`**

Specifies the Script trigger type. Depending upon the value of this property, additional properties may need to be configured to fully configure the trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ScriptTriggerType
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggers:Script:Trigger Type**
- C Attribute: **NIFGEN_ATTR_SCRIPT_TRIGGER_TYPE**

serial_number

`nifgen.Session.`**`serial_number`**

The signal generator's serial number.

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: **Instrument:Serial Number**
- C Attribute: **NIFGEN_ATTR_SERIAL_NUMBER**

simulate

`nifgen.Session.`**`simulate`**

Specifies whether to simulate NI-FGEN I/O operations. If simulation is enabled, NI-FGEN methods perform range checking and call Ivi_GetAttribute and Ivi_SetAttribute, but they do not perform device I/O. For output parameters that represent device data, NI-FGEN methods return calculated values. Default Value: False Use `nifgen.Session.InitWithOptions()` to override default value.

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	bool
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Instrument:Inherent IVI Attributes:User Options:Simulate**
- C Attribute: **NIFGEN_ATTR_SIMULATE**

specific_driver_description

`nifgen.Session.`**`specific_driver_description`**

Returns a brief description of NI-FGEN.

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: **Instrument:Inherent IVI Attributes:Driver Identification:Description**
- C Attribute: **NIFGEN_ATTR_SPECIFIC_DRIVER_DESCRIPTION**

major_version

`nifgen.Session.`**`major_version`**

Returns the major version number of NI-FGEN.

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: **Instrument:Obsolete:Major Version**
- C Attribute: **NIFGEN_ATTR_SPECIFIC_DRIVER_MAJOR_VERSION**

minor_version

`nifgen.Session.`**`minor_version`**

Returns the minor version number of NI-FGEN.

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: **Instrument:Obsolete:Minor Version**
- C Attribute: **NIFGEN_ATTR_SPECIFIC_DRIVER_MINOR_VERSION**

specific_driver_revision

`nifgen.Session.`**`specific_driver_revision`**

A string that contains additional version information about NI-FGEN.

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: **Instrument:Inherent IVI Attributes:Driver Identification:Revision**
- C Attribute: **NIFGEN_ATTR_SPECIFIC_DRIVER_REVISION**

specific_driver_vendor

`nifgen.Session.`**`specific_driver_vendor`**

A string that contains the name of the vendor that supplies NI-FGEN.

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: **Instrument:Inherent IVI Attributes:Driver Identification:Driver Vendor**
- C Attribute: **NIFGEN_ATTR_SPECIFIC_DRIVER_VENDOR**

started_event_output_terminal

`nifgen.Session.`**`started_event_output_terminal`**

Specifies the destination terminal for the Started 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: **Events:Started:Output Terminal**
- C Attribute: **NIFGEN_ATTR_STARTED_EVENT_OUTPUT_TERMINAL**

start_trigger_type

`nifgen.Session.`**`start_trigger_type`**

Specifies whether you want the Start trigger to be a Digital Edge, or Software trigger. You can also choose None as the value for this property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.StartTriggerType
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggers:Start:Trigger Type**
- C Attribute: **NIFGEN_ATTR_START_TRIGGER_TYPE**

streaming_space_available_in_waveform

`nifgen.Session.`**`streaming_space_available_in_waveform`**

Indicates the space available (in samples) in the streaming waveform for writing new data. During generation, this available space may be in multiple locations with, for example, part of the available space at the end of the streaming waveform and the rest at the beginning. In this situation, writing a block of waveform data the size of the total space available in the streaming waveform causes NI-FGEN to return an error, as NI-FGEN will not wrap the data from the end of the waveform to the beginning and cannot write data past the end of the waveform buffer. To avoid writing data past the end of the waveform, write new data to the waveform in a fixed size that is an integer divisor of the total size of the streaming waveform. Used in conjunction with the *`nifgen.Session.streaming_waveform_handle`* or *`nifgen.Session.streaming_waveform_name`* properties.

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: **Arbitrary Waveform:Data Transfer:Streaming:Space Available in Streaming Waveform**
- C Attribute: **NIFGEN_ATTR_STREAMING_SPACE_AVAILABLE_IN_WAVEFORM**

streaming_waveform_handle

`nifgen.Session.`**`streaming_waveform_handle`**

Specifies the waveform handle of the waveform used to continuously stream data during generation. This property defaults to -1 when no streaming waveform is specified. Used in conjunction with *`nifgen.Session.streaming_space_available_in_waveform`*.

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: **Arbitrary Waveform:Data Transfer:Streaming:Streaming Waveform Handle**
- C Attribute: **NIFGEN_ATTR_STREAMING_WAVEFORM_HANDLE**

streaming_waveform_name

`nifgen.Session.`**`streaming_waveform_name`**

Specifies the name of the waveform used to continuously stream data during generation. This property defaults to // when no streaming waveform is specified. Use in conjunction with *`nifgen.Session.streaming_space_available_in_waveform`*.

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: **Arbitrary Waveform:Data Transfer:Streaming:Streaming Waveform Name**
- C Attribute: **NIFGEN_ATTR_STREAMING_WAVEFORM_NAME**

streaming_write_timeout

`nifgen.Session.`**`streaming_write_timeout`**

Specifies the maximum amount of time allowed to complete a streaming write operation.

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: **Arbitrary Waveform:Data Transfer:Streaming:Streaming Write Timeout**
- C Attribute: **NIFGEN_ATTR_STREAMING_WRITE_TIMEOUT**

supported_instrument_models

`nifgen.Session.`**`supported_instrument_models`**

Returns a model code of the device. For NI-FGEN versions that support more than one device, this property contains a comma-separated list of supported device models.

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: **Instrument:Inherent IVI Attributes:Driver Capabilities:Supported Instrument Models**
- C Attribute: **NIFGEN_ATTR_SUPPORTED_INSTRUMENT_MODELS**

terminal_configuration

`nifgen.Session.`**`terminal_configuration`**

Specifies whether gain and offset values will be analyzed based on single-ended or differential operation.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TerminalConfiguration
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Output:Terminal Configuration**
- C Attribute: **NIFGEN_ATTR_TERMINAL_CONFIGURATION**

trigger_mode

`nifgen.Session.`**`trigger_mode`**

Controls the trigger mode.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TriggerMode
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Triggers:Trigger Mode**
- C Attribute: **NIFGEN_ATTR_TRIGGER_MODE**

wait_behavior

`nifgen.Session.`**`wait_behavior`**

Specifies the behavior of the output while waiting for a script trigger or during a wait instruction. The output can be configured to hold the last generated voltage before waiting or jump to the Wait Value.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.WaitBehavior
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Output:Advanced:Wait Behavior**
- C Attribute: **NIFGEN_ATTR_WAIT_BEHAVIOR**

wait_value

`nifgen.Session.`**`wait_value`**

Specifies the value to generate while waiting. The Wait Behavior must be configured to jump to this value.

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: **Output:Advanced:Wait Value**

- C Attribute: **NIFGEN_ATTR_WAIT_VALUE**

waveform_quantum

`nifgen.Session.`**`waveform_quantum`**

The size of each arbitrary waveform must be a multiple of a quantum value. This property returns the quantum value that the signal generator allows. For example, when this property returns a value of 8, all waveform sizes must be a multiple of 8. Typically, this value is constant for the signal generator.

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: **Arbitrary Waveform:Capabilities:Waveform Quantum**
- C Attribute: **NIFGEN_ATTR_WAVEFORM_QUANTUM**

NI-TClk Support

`nifgen.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

- *data_transfer_maximum_in_flight_reads*
- *data_transfer_preferred_packet_size*
- *digital_data_mask*
- *digital_edge_script_trigger_edge*
- *digital_edge_script_trigger_source*
- *digital_edge_start_trigger_edge*
- *digital_edge_start_trigger_source*
- *digital_filter_enabled*
- *digital_filter_interpolation_factor*
- *digital_gain*
- *digital_pattern_enabled*
- *digital_static_value*
- *done_event_output_terminal*
- *driver_setup*
- *exported_onboard_reference_clock_output_terminal*
- *exported_reference_clock_output_terminal*
- *exported_sample_clock_divisor*
- *exported_sample_clock_output_terminal*
- *exported_sample_clock_timebase_divisor*
- *exported_sample_clock_timebase_output_terminal*
- *exported_script_trigger_output_terminal*
- *exported_start_trigger_output_terminal*
- *external_clock_delay_binary_value*
- *external_sample_clock_multiplier*
- *file_transfer_block_size*
- *filter_correction_frequency*
- *flatness_correction_enabled*
- *fpga_bitfile_path*
- *freq_list_duration_quantum*
- *freq_list_handle*
- *func_amplitude*
- *func_buffer_size*
- *func_dc_offset*
- *func_duty_cycle_high*
- *func_frequency*

- *func_max_buffer_size*
- *func_start_phase*
- *func_waveform*
- *idle_behavior*
- *idle_value*
- *instrument_firmware_revision*
- *instrument_manufacturer*
- *instrument_model*
- *io_resource_descriptor*
- *load_impedance*
- *logical_name*
- *marker_events_count*
- *marker_event_output_terminal*
- *max_freq_list_duration*
- *max_freq_list_length*
- *max_loop_count*
- *max_num_freq_lists*
- *max_num_sequences*
- *max_num_waveforms*
- *max_sequence_length*
- *max_waveform_size*
- *memory_size*
- *min_freq_list_duration*
- *min_freq_list_length*
- *min_sequence_length*
- *min_waveform_size*
- *module_revision*
- *channel_count*
- *output_enabled*
- *output_impedance*
- *output_mode*
- *ready_for_start_event_output_terminal*
- *reference_clock_source*
- *ref_clock_frequency*
- *sample_clock_source*

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 `niFgen_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

nifgen.Session.channels[]

```
session.channels['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

script_triggers

nifgen.Session.script_triggers[]

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.script_triggers['0-2'].channel_enabled = True
```

passes a string of `'ScriptTrigger0, ScriptTrigger1, ScriptTrigger2'` to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.script_triggers['ScriptTrigger0-ScriptTrigger2'].channel_enabled␣
↪= True
```

passes a string of `'ScriptTrigger0, ScriptTrigger1, ScriptTrigger2'` to the set attribute function.

markers

nifgen.Session.markers[]

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.markers['0-2'].channel_enabled = True
```

passes a string of `'Marker0, Marker1, Marker2'` to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.markers['Marker0-Marker2'].channel_enabled = True
```

passes a string of `'Marker0, Marker1, Marker2'` to the set attribute function.

Enums

Enums used in NI-FGEN

AnalogPath

class nifgen.**AnalogPath**

MAIN
Specifies use of the main path. NI-FGEN chooses the amplifier based on the user-specified gain.

DIRECT
Specifies use of the direct path.

FIXED_LOW_GAIN
Specifies use of the low-gain amplifier in the main path, no matter what value the user specifies for gain. This setting limits the output range.

FIXED_HIGH_GAIN
Specifies use of the high-gain amplifier in the main path.

BusType

class nifgen.**BusType**

INVALID
Indicates an invalid bus type.

AT
Indicates the signal generator is the AT bus type.

PCI
Indicates the signal generator is the PCI bus type.

PXI
Indicates the signal generator is the PXI bus type.

VXI
Indicates the signal generator is the VXI bus type.

PCMCIA
Indicates the signal generator is the PCI-CMA bus type.

PXIE
Indicates the signal generator is the PXI Express bus type.

ByteOrder

class nifgen.**ByteOrder**

LITTLE

BIG

ClockMode

class nifgen.**ClockMode**

HIGH_RESOLUTION
High resolution sampling—Sample rate is generated by a high–resolution clock source.

DIVIDE_DOWN
Divide down sampling—Sample rates are generated by dividing the source frequency.

AUTOMATIC
Automatic Selection—NI-FGEN selects between the divide–down and high–resolution clocking modes.

DataMarkerEventLevelPolarity

class nifgen.**DataMarkerEventLevelPolarity**

HIGH
When the operation is ready to start, the Ready for Start event level is high.

LOW
When the operation is ready to start, the Ready for Start event level is low.

HardwareState

class nifgen.**HardwareState**

IDLE

WAITING_FOR_START_TRIGGER

RUNNING

DONE

HARDWARE_ERROR

IdleBehavior

class nifgen.**IdleBehavior**

HOLD_LAST
While in an Idle or Wait state, the output signal remains at the last voltage generated prior to entering the state.

JUMP_TO
While in an Idle or Wait state, the output signal remains at the value configured in the Idle or Wait value property.

OutputMode

class nifgen.**OutputMode**

FUNC
Standard Method mode— Generates standard method waveforms such as sine, square, triangle, and so on.

ARB
Arbitrary waveform mode—Generates waveforms from user-created/provided waveform arrays of numeric data.

SEQ
Arbitrary sequence mode — Generates downloaded waveforms in an order your specify.

FREQ_LIST
Frequency List mode—Generates a standard method using a list of frequencies you define.

SCRIPT
Script mode—Allows you to use scripting to link and loop multiple waveforms in complex combinations.

ReferenceClockSource

`class nifgen.ReferenceClockSource`

CLOCK_IN
Specifies that the CLK IN input signal from the front panel connector is used as the Reference Clock source.

NONE
Specifies that a Reference Clock is not used.

ONBOARD_REFERENCE_CLOCK
Specifies that the onboard Reference Clock is used as the Reference Clock source.

PXI_CLOCK
Specifies the PXI Clock is used as the Reference Clock source.

RTSI_7
Specifies that the RTSI line 7 is used as the Reference Clock source.

RelativeTo

`class nifgen.RelativeTo`

START

CURRENT

SampleClockSource

`class nifgen.SampleClockSource`

CLOCK_IN
Specifies that the signal at the CLK IN front panel connector is used as the Sample Clock source.

DDC_CLOCK_IN
Specifies that the Sample Clock from DDC connector is used as the Sample Clock source.

ONBOARD_CLOCK
Specifies that the onboard clock is used as the Sample Clock source.

PXI_STAR_LINE
Specifies that the PXI_STAR trigger line is used as the Sample Clock source.

PXI_TRIGGER_LINE_0_RTSI_0
Specifies that the PXI or RTSI line 0 is used as the Sample Clock source.

PXI_TRIGGER_LINE_1_RTSI_1
Specifies that the PXI or RTSI line 1 is used as the Sample Clock source.

PXI_TRIGGER_LINE_2_RTSI_2
Specifies that the PXI or RTSI line 2 is used as the Sample Clock source.

PXI_TRIGGER_LINE_3_RTSI_3
Specifies that the PXI or RTSI line 3 is used as the Sample Clock source.

PXI_TRIGGER_LINE_4_RTSI_4
Specifies that the PXI or RTSI line 4 is used as the Sample Clock source.

PXI_TRIGGER_LINE_5_RTSI_5
Specifies that the PXI or RTSI line 5 is used as the Sample Clock source.

PXI_TRIGGER_LINE_6_RTSI_6
Specifies that the PXI or RTSI line 6 is used as the Sample Clock source.

PXI_TRIGGER_LINE_7_RTSI_7
Specifies that the PXI or RTSI line 7 is used as the Sample Clock source.

SampleClockTimebaseSource

class nifgen.**SampleClockTimebaseSource**

CLOCK_IN
Specifies that the external signal on the CLK IN front panel connector is used as the source.

ONBOARD_CLOCK
Specifies that the onboard Sample Clock timebase is used as the source.

ScriptTriggerDigitalEdgeEdge

class nifgen.**ScriptTriggerDigitalEdgeEdge**

RISING
Rising Edge

FALLING
Falling Edge

ScriptTriggerType

class nifgen.**ScriptTriggerType**

TRIG_NONE
No trigger is configured. Signal generation starts immediately.

DIGITAL_EDGE
Trigger is asserted when a digital edge is detected.

DIGITAL_LEVEL
Trigger is asserted when a digital level is detected.

SOFTWARE_EDGE
Trigger is asserted when a software edge is detected.

StartTriggerDigitalEdgeEdge

class nifgen.**StartTriggerDigitalEdgeEdge**

RISING
Rising Edge

FALLING
Falling Edge

StartTriggerType

class nifgen.**StartTriggerType**

TRIG_NONE
None

DIGITAL_EDGE
Digital Edge

SOFTWARE_EDGE
Software Edge

P2P_ENDPOINT_FULLNESS
P2P Endpoint Fullness

TerminalConfiguration

class nifgen.**TerminalConfiguration**

SINGLE_ENDED
Single-ended operation

DIFFERENTIAL
Differential operation

Trigger

class nifgen.**Trigger**

START

SCRIPT

TriggerMode

class nifgen.**TriggerMode**

SINGLE
Single Trigger Mode - The waveform you describe in the sequence list is generated only once by going through the entire staging list. Only one trigger is required to start the waveform generation. You can use Single trigger mode with the output mode in any mode. After a trigger is received, the waveform generation starts from the first stage and continues through to the last stage. Then, the last stage generates repeatedly until you stop the waveform generation.

CONTINUOUS
Continuous Trigger Mode - The waveform you describe in the staging list generates infinitely by repeatedly cycling through the staging list. After a trigger is received, the waveform generation starts from the first stage and continues through to the last stage. After the last stage completes, the waveform generation loops back to the start of the first stage and continues until it is stopped. Only one trigger is required to start the waveform generation.

STEPPED
Stepped Trigger Mode - After a start trigger is received, the waveform described by the first stage generates. Then, the device waits for the next trigger signal. On the next trigger, the waveform described by the second stage generates, and so on. After the staging list completes, the waveform generation returns to the first stage and continues in a cyclic fashion. After any stage has generated completely, the first eight samples of the next stage are repeated continuously until the next trigger is received. trigger mode.

Note: In Frequency List mode, Stepped trigger mode is the same as Burst

BURST
Burst Trigger Mode - After a start trigger is received, the waveform described by the first stage generates until another trigger is received. At the next trigger, the buffer of the previous stage completes, and then the waveform described by the second stage generates. After the staging list completes, the waveform generation returns to the first stage and continues in a cyclic fashion. In Frequency List mode, the duration instruction is ignored, and the trigger switches the frequency to the next frequency in the list. trigger mode.

Note: In Frequency List mode, Stepped trigger mode is the same as Burst

WaitBehavior

class nifgen.**WaitBehavior**

HOLD_LAST
While in an Idle or Wait state, the output signal remains at the last voltage generated prior to entering the state.

JUMP_TO
While in an Idle or Wait state, the output signal remains at the value configured in the Idle or Wait value property.

Waveform

class nifgen.**Waveform**

SINE
Sinusoid waveform

SQUARE
Square waveform

TRIANGLE
Triange waveform

RAMP_UP
Positive ramp waveform

RAMP_DOWN
Negative ramp waveform

DC
Constant voltage

NOISE
White noise

USER
User-defined waveform as defined by the *nifgen.Session.define_user_standard_waveform()* method.

Exceptions and Warnings

Error

exception nifgen.errors.**Error**
Base exception type that all NI-FGEN exceptions derive from

DriverError

exception nifgen.errors.**DriverError**
An error originating from the NI-FGEN driver

UnsupportedConfigurationError

exception nifgen.errors.**UnsupportedConfigurationError**
An error due to using this module in an usupported platform.

DriverNotInstalledError

exception nifgen.errors.**DriverNotInstalledError**
An error due to using this module without the driver runtime installed.

InvalidRepeatedCapabilityError

exception nifgen.errors.**InvalidRepeatedCapabilityError**
An error due to an invalid character in a repeated capability

SelfTestError

exception nifgen.errors.**SelfTestError**
An error due to a failed self-test

DriverWarning

exception nifgen.errors.**DriverWarning**
A warning originating from the NI-FGEN driver

Examples

You can download all nifgen examples here

nifgen_arb_waveform.py

Listing 10: (nifgen_arb_waveform.py)

```python
#!/usr/bin/python

import argparse
import math
import nifgen
import sys
import time


def create_waveform_data(number_of_samples):
    waveform_data = []
    angle_per_sample = (2 * math.pi) / number_of_samples
    for i in range(number_of_samples):
        waveform_data.append(math.sin(i * angle_per_sample) * math.sin(i * angle_per_
↪sample * 20))
    return waveform_data


def example(resource_name, options, samples, gain, offset, gen_time):
    waveform_data = create_waveform_data(samples)
    with nifgen.Session(resource_name=resource_name, options=options) as session:
        session.output_mode = nifgen.OutputMode.ARB
        waveform = session.create_waveform(waveform_data_array=waveform_data)
        session.configure_arb_waveform(waveform_handle=waveform, gain=gain,␣
↪offset=offset)
        with session.initiate():
            time.sleep(gen_time)

```

(continues on next page)

(continued from previous page)

```python

def _main(argsv):
    parser = argparse.ArgumentParser(description='Continuously generates an arbitrary␣
↪waveform.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of a National Instruments Arbitrary Waveform Generator')
    parser.add_argument('-s', '--samples', default=100000, type=int, help='Number of␣
↪samples')
    parser.add_argument('-g', '--gain', default=1.0, type=float, help='Gain')
    parser.add_argument('-o', '--offset', default=0.0, type=float, help='DC offset (V)
↪')
    parser.add_argument('-t', '--time', default=5.0, type=float, help='Generation␣
↪time (s)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.samples, args.gain, args.
↪offset, args.time)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5433 (2CH)', 'BoardType':
↪'PXIe', }, }
    example('PXI1Slot2', options, 100000, 1.0, 0.0, 5.0)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5433 (2CH);
↪BoardType:PXIe', ]
    _main(cmd_line)


if __name__ == '__main__':
    main()


```

nifgen_script.py

Listing 11: (nifgen_script.py)

```python
#!/usr/bin/python

import argparse
import nifgen
import numpy as np
from scipy import signal
import sys
import time

```

(continues on next page)

(continued from previous page)

```python
number_of_points = 256


def calculate_sinewave():
    time = np.linspace(start=0, stop=10, num=number_of_points)     # np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
    amplitude = np.sin(time)
    sinewave = amplitude.tolist()                                # List of Float
    return sinewave


def calculate_rampup():
    ramp = np.linspace(start=0, stop=0.5, num=number_of_points)    # np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
    ramp_up = ramp.tolist()                                      # List of Float
    return ramp_up


def calculate_rampdown():
    ramp = np.linspace(start=0, stop=0.5, num=number_of_points)    # np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
    ramp_down = ramp.tolist()                                    # List of Float
    ramp_down.reverse()                                          # Reverse list to get a ramp down
    return ramp_down


def calculate_square():
    time = np.linspace(start=0, stop=10, num=number_of_points)     # np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
    square_build = signal.square(t=time, duty=0.5)               # signal.square(t, duty=0.5)
    square = square_build.tolist()                               # List of Float
    return square


def calculate_triangle():
    time = np.linspace(start=0, stop=1, num=number_of_points)      # np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
    triangle_build = signal.sawtooth(t=time)                     # signal.sawtooth(t, width=1)
    triangle = triangle_build.tolist()                           # List of Float
    return triangle


def calculate_gaussian_noise():
    random_noise = np.random.normal(loc=0, scale=0.1, size=number_of_points)   # random.normal(loc=0.0, scale=1.0, size=None)
    noise = random_noise.tolist()                                            # List of Float
    return noise


SCRIPT_ALL = '''
script scriptmulti
  repeat until scriptTrigger0
    generate rampup
```

(continues on next page)

(continued from previous page)

```
    generate sine
    generate rampdown
  end repeat
  repeat until scriptTrigger0
    generate rampdown
    generate square
    generate rampup
  end repeat
  repeat until scriptTrigger0
    generate rampup
    generate rampdown
  end repeat
  repeat until scriptTrigger0
    generate sine
  end repeat
  repeat until scriptTrigger0
    generate triangle
  end repeat
  repeat until scriptTrigger0
    generate rampdown
    generate noise
    generate rampup
  end repeat
end script

script scriptsine
  repeat until scriptTrigger0
    generate sine
  end repeat
end script

script scriptrampup
  repeat until scriptTrigger0
    generate rampup
  end repeat
end script

script scriptrampdown
  repeat until scriptTrigger0
    generate rampdown
  end repeat
end script

script scriptsquare
  repeat until scriptTrigger0
    generate square
  end repeat
end script

script scripttriangle
  repeat until scriptTrigger0
    generate triangle
  end repeat
end script

script scriptnoise
  repeat until scriptTrigger0
```

(continues on next page)

(continued from previous page)

```
    generate noise
  end repeat
end script
'''


def example(resource_name, options, shape, channel):
    with nifgen.Session(resource_name=resource_name, options=options, channel_
↪name=channel) as session:
        # CONFIGURATION
        # 1 - Set the mode to Script
        session.output_mode = nifgen.OutputMode.SCRIPT

        # 2 - Configure Trigger:
        # SOFTWARE TRIGGER: used in the script
        session.script_triggers[0].script_trigger_type = nifgen.ScriptTriggerType.
↪SOFTWARE_EDGE  # TRIG_NONE / DIGITAL_EDGE / DIGITAL_LEVEL / SOFTWARE_EDGE
        session.script_triggers[0].digital_edge_script_trigger_edge = nifgen.
↪ScriptTriggerDigitalEdgeEdge.RISING  # RISING / FAILING

        # 3 - Calculate and write different waveform data to the device's onboard␣
↪memory
        session.channels[channel].write_waveform('sine', calculate_sinewave())
↪# (waveform_name, data)
        session.channels[channel].write_waveform('rampup', calculate_rampup())
        session.channels[channel].write_waveform('rampdown', calculate_rampdown())
        session.channels[channel].write_waveform('square', calculate_square())
        session.channels[channel].write_waveform('triangle', calculate_triangle())
        session.channels[channel].write_waveform('noise', calculate_gaussian_noise())

        # 4 - Script to generate
        # supported shapes: SINE / SQUARE / TRIANGLE / RAMPUP / RAMPDOWN / NOISE /␣
↪MULTI
        script_name = 'script{}'.format(shape.lower())
        num_triggers = 6 if shape.upper() == 'MULTI' else 1  # Only multi needs two␣
↪triggers, all others need one

        session.channels[channel].write_script(SCRIPT_ALL)
        session.script_to_generate = script_name

        # LAUNCH
        with session.initiate():
            for x in range(num_triggers):
                time.sleep(10)
                session.script_triggers[0].send_software_edge_trigger()


def _main(argsv):
    parser = argparse.ArgumentParser(description='Generate different shape waveforms.
↪', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of a National Instruments Arbitrary Waveform Generator')
    parser.add_argument('-s', '--shape', default='SINE', help='Shape of the signal to␣
↪generate')
    parser.add_argument('-c', '--channel', default='0', help='Channel to use when␣
↪generating')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
```

(continues on next page)

(continued from previous page)

```
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.shape.upper(), args.channel)


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5433 (2CH)', 'BoardType':
↪'PXIe', }, }
    example('PXI1Slot2', options, 'SINE', '0')


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5433 (2CH);
↪BoardType:PXIe', '--channel', '0', ]
    _main(cmd_line)


def main():
    _main(sys.argv[1:])


if __name__ == '__main__':
    main()




```

nifgen_standard_function.py

Listing 12: (nifgen_standard_function.py)

```
#!/usr/bin/python

import argparse
import nifgen
import sys
import time


def example(resource_name, options, waveform, frequency, amplitude, offset, phase,␣
↪gen_time):
    with nifgen.Session(resource_name=resource_name, options=options) as session:
        session.output_mode = nifgen.OutputMode.FUNC
        session.configure_standard_waveform(waveform=nifgen.Waveform[waveform],␣
↪amplitude=amplitude, frequency=frequency, dc_offset=offset, start_phase=phase)
        with session.initiate():
            time.sleep(gen_time)


def _main(argsv):
    supported_waveforms = list(nifgen.Waveform.__members__.keys())[:-1]  # no support␣
↪for user-defined waveforms in example
    parser = argparse.ArgumentParser(description='Generates the standard function.',␣
↪formatter_class=argparse.ArgumentDefaultsHelpFormatter)
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of a National Instruments Function Generator')
    parser.add_argument('-w', '--waveform', default=supported_waveforms[0],␣
↪choices=supported_waveforms, type=str.upper, help='Standard waveform')
    parser.add_argument('-f', '--frequency', default=1000, type=float, help=
↪'Frequency (Hz)')
    parser.add_argument('-a', '--amplitude', default=1.0, type=float, help='Amplitude␣
↪(Vpk-pk)')
    parser.add_argument('-o', '--offset', default=0.0, type=float, help='DC offset (V)
↪')
    parser.add_argument('-p', '--phase', default=0.0, type=float, help='Start phase␣
↪(deg)')
    parser.add_argument('-t', '--time', default=5.0, type=float, help='Generation␣
↪time (s)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.option_string, args.waveform, args.frequency,␣
↪args.amplitude, args.offset, args.phase, args.time)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5433 (2CH)', 'BoardType':
↪'PXIe', }, }
    example('PXI1Slot2', options, 'SINE', 1000, 1.0, 0.0, 0.0, 5.0)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5433 (2CH);
↪BoardType:PXIe', ]
    _main(cmd_line)


if __name__ == '__main__':
    main()



```

nifgen_trigger.py

Listing 13: (nifgen_trigger.py)

```
import argparse
import nifgen
import sys
import time


def example(resource_name1, resource_name2, options, waveform, gen_time):
```

(continues on next page)

(continued from previous page)

```
    with nifgen.Session(resource_name=resource_name1, options=options) as session1,␣
↪nifgen.Session(resource_name=resource_name2, options=options) as session2:
        session_list = [session1, session2]
        for session in session_list:
            session.output_mode = nifgen.OutputMode.FUNC
            session.configure_standard_waveform(waveform=nifgen.Waveform[waveform],␣
↪amplitude=1.0, frequency=1000, dc_offset=0.0, start_phase=0.0)
        session1.start_trigger_type = nifgen.StartTriggerType.SOFTWARE_EDGE
        session2.start_trigger_type = nifgen.StartTriggerType.DIGITAL_EDGE
        session2.digital_edge_start_trigger_edge = nifgen.StartTriggerDigitalEdgeEdge.
↪RISING
        session2.digital_edge_start_trigger_source = '/' + resource_name1 + '/0/
↪StartTrigger'
        with session2.initiate():
            with session1.initiate():
                session1.send_software_edge_trigger(nifgen.Trigger.START)
                time.sleep(gen_time)


def _main(argsv):
    supported_waveforms = list(nifgen.Waveform.__members__.keys())[:-1]  # no support␣
↪for user-defined waveforms in example
    parser = argparse.ArgumentParser(description='Triggers one device on the start␣
↪trigger of another device.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n1', '--resource-name1', default='PXI1Slot2', help=
↪'Resource name of a NI Function Generator')
    parser.add_argument('-n2', '--resource-name2', default='PXI1Slot3', help=
↪'Resource name of a NI Function Generator')
    parser.add_argument('-w', '--waveform', default=supported_waveforms[0],␣
↪choices=supported_waveforms, type=str.upper, help='Standard waveform')
    parser.add_argument('-t', '--time', default=5.0, type=float, help='Generation␣
↪time (s)')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option␣
↪string')
    args = parser.parse_args(argsv)
    example(args.resource_name1, args.resource_name2, args.option_string, args.
↪waveform, args.time)


def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5433 (2CH)', 'BoardType':
↪'PXIe', }, }
    example('PXI1Slot2', 'PXI1Slot3', options, 'SINE', 5.0)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5433 (2CH);
↪BoardType:PXIe', ]
    _main(cmd_line)


if __name__ == '__main__':
    main()
```

7.5 niscope module

7.5.1 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~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install niscope
```

7.5.2 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']

    # Find all record number 3
    rec3 = [wfm for wfm in waveforms if wfm.record == 3]
```

The waveform returned from fetch is a flat list of Python objects

- Attributes:
  - **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 = binary data * gain factor + offset

- **offset** (float) the offset factor of the given channel; useful for scaling binary data with the following formula:

    voltage = binary data * gain factor + offset

- **samples** (array of float) floating point array of samples. Length will be of the actual samples acquired

- Such that all record 0 waveforms are first. For example, with a channel list of 0,1, you would have the following index values:
  - index 0 = record 0, channel 0
  - index 1 = record 0, channel 1
  - index 2 = record 1, channel 0
  - index 3 = record 1, channel 1
  - etc.

If you need more performance or need to work with SciPy, you can use the *fetch_into()* method instead of *fetch()*. This method takes an already allocated numpy array and puts the acquired samples in it. Data types supported:

- *numpy.float64*
- *numpy.int8*
- *numpy.in16*
- *numpy.int32*

```
voltage_range = 1.0
record_length = 2000
channels = [0, 1]
num_channels = len(channels)
num_records = 5
wfm = numpy.ndarray(num_channels * record_length, dtype=numpy.int8)
session.configure_vertical(voltage_range, niscope.VerticalCoupling.AC)
session.configure_horizontal_timing(50000000, record_length, 50.0, num_records, True)
with session.initiate():
    waveform_infos = session.channels[channels].fetch_into(wfm=wfm, num_records=num_
↪records)
```

The waveform_infos returned from fetch_into is a 1D list of Python objects

- Attributes:
  - **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 asquire 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 = binary data * gain factor + offset

  - **offset** (float) the offset factor of the given channel; useful for scaling binary data with the following formula:

    voltage = binary data * gain factor + offset

- **samples** (numpy array of datatype used) floating point array of samples. Length will be of the actual samples acquired

Note: Python 3 only

- Such that all record 0 waveforms are first. For example, with a channel list of 0,1, you would have the following index values:
  - index 0 = record 0, channel 0
  - index 1 = record 0, channel 1
  - index 2 = record 1, channel 0
  - index 3 = record 1, channel 1
  - etc.

Note: When using Python 2, the waveform_infos objects do not include the waveform for that record. Instead, samples are in the waveform passed into the function using the following layout:

- index 0 = record 0, channel 0
- index *x* = record 0, channel 1
- index 2*x* = record 1, channel 0
- index 3*x* = record 1, channel 1
- etc.
- Where *x* = the record length

Additional examples for NI-SCOPE are located in src/niscope/examples/ directory.

7.5.3 API Reference

Session

class `niscope.`**`Session`**(*self*, *resource_name*, *id_query=False*, *reset_device=False*, *options={}*)

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	{}

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.ClearableMe*

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 detect-

ing 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 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.
- **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 detect-

ing 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_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

- *Session*

- *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*

- *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*

- *NI-TClk Support*

Repeated Capabilities

Repeated capabilities attributes are used to set the *channel_string* parameter to the underlying driver function call. This can be the actual function based on the `Session` method being called, or it can be the appropriate Get/Set Attribute function, such as `niScope_SetAttributeViInt32()`.

Repeated capabilities attributes use the indexing operator `[]` to indicate the repeated capabilities. The parameter can be a string, list, tuple, or slice (range). Each element of those can be a string or an integer. If it is a string, you can indicate a range using the same format as the driver: `'0-2'` or `'0:2'`

Some repeated capabilities use a prefix before the number and this is optional

channels

`niscope.Session.channels[]`

```
session.channels['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

instruments

`niscope.Session.instruments[]`

```
session.instruments['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

Enums

Enums used in NI-SCOPE

AcquisitionStatus

class `niscope.`**`AcquisitionStatus`**

`COMPLETE`

`IN_PROGRESS`

`STATUS_UNKNOWN`

AcquisitionType

class 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.

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

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 14: (niscope_fetch.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):
    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 a National Instruments Digitizer')
```

(continues on next page)

(continued from previous page)

```
    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()

```

niscope_fetch_forever.py

Listing 15: (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):
```

(continues on next page)

(continued from previous page)

```
    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()
        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 a National Instruments 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)


```

(continues on next page)

(continued from previous page)

```python
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)


if __name__ == '__main__':
    main()

```

niscope_read.py

Listing 16: (niscope_read.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)
        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 a National Instruments 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')
```

(continues on next page)

(continued from previous page)

```python
    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()

```

7.6 niswitch module

7.6.1 Installation

As a prerequisite to using the niswitch module, you must install the NI-SWITCH runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

The nimi-python modules (i.e. for **NI-SWITCH**) can be installed with pip:

```
$ python -m pip install niswitch~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install niswitch
```

7.6.2 Usage

The following is a basic example of using the **niswitch** module to open a session to a Switch and connect channels.

```python
import niswitch
with niswitch.Session("Dev1") as session:
    session.connect(channel1='r0', channel2='c0')
```

Additional examples for NI-SWITCH are located in src/niswitch/examples/ directory.

7.6.3 API Reference

Session

class niswitch.**Session**(*self*, *resource_name*, *topology="Configured Topology"*, *simulate=False*, *reset_device=False*)

Returns a session handle used to identify the switch in all subsequent instrument driver calls and sets the topology of the switch. `niswitch.Session.__init__()` creates a new IVI instrument driver session for the switch specified in the resourceName parameter. The driver uses the topology specified in the topology parameter and overrides the topology specified in MAX. Note: When initializing an NI SwitchBlock device with topology, you must specify the toplogy created when you configured the device in MAX, using either NISWITCH_TOPOLOGY_CONFIGURED_TOPOLOGY or the toplogy string of the device. Refer to the Initializing with Toplogy for NI SwitchBlock Devices topic in the NI Switches Help for information about determining the topology string of an NI SwitchBlock device. By default, the switch is reset to a known state. Enable simulation by specifying the topology and setting the simulate parameter to True.

Parameters

- **resource_name** (*str*) – Resource name of the switch module to initialize. Default value: None Syntax: Optional fields are shown in square brackets ([]). Configured in MAX Under Valid Syntax Devices and Interfaces DeviceName Traditional NI-DAQ Devices SCXI[chassis ID]::slot number PXI System PXI[bus number]::device number TIP: IVI logical names are also valid for the resource name. Default values for optional fields: chassis ID = 1 bus number = 0 Example resource names: Resource Name Description SC1Mod3 NI-DAQmx module in chassis "SC1" slot 3 MySwitch NI-DAQmx module renamed to "MySwitch" SCXI1::3 Traditional NI-DAQ module in chassis 1, slot 3 SCXI::3 Traditional NI-DAQ module in chassis 1, slot 3 PXI0::16 PXI bus 0, device number 16 PXI::16 PXI bus 0, device number 16
- **topology** (*str*) – Pass the topology name you want to use for the switch you specify with Resource Name parameter. You can also pass NISWITCH_TOPOLOGY_CONFIGURED_TOPOLOGY to use the last topology that was configured for the device in MAX. Default Value: NISWITCH_TOPOLOGY_CONFIGURED_TOPOLOGY Valid Values: NISWITCH_TOPOLOGY_1127_1_WIRE_64X1_MUX NISWITCH_TOPOLOGY_1127_2_WIRE_32X1_MUX NISWITCH_TOPOLOGY_1127_2_WIRE_4X8_MATR
NISWITCH_TOPOLOGY_1127_4_WIRE_16X1_MUX NISWITCH_TOPOLOGY_1127_INDEPENDENT
NISWITCH_TOPOLOGY_1128_1_WIRE_64X1_MUX NISWITCH_TOPOLOGY_1128_2_WIRE_32X1_MUX
NISWITCH_TOPOLOGY_1128_2_WIRE_4X8_MATRIX
NISWITCH_TOPOLOGY_1128_4_WIRE_16X1_MUX NISWITCH_TOPOLOGY_1128_INDEPENDENT
NISWITCH_TOPOLOGY_1129_2_WIRE_16X16_MATRIX
NISWITCH_TOPOLOGY_1129_2_WIRE_8X32_MATRIX
NISWITCH_TOPOLOGY_1129_2_WIRE_4X64_MATRIX
NISWITCH_TOPOLOGY_1129_2_WIRE_DUAL_8X16_MATRIX
NISWITCH_TOPOLOGY_1129_2_WIRE_DUAL_4X32_MATRIX
NISWITCH_TOPOLOGY_1129_2_WIRE_QUAD_4X16_MATRIX
NISWITCH_TOPOLOGY_1130_1_WIRE_256X1_MUX NISWITCH_TOPOLOGY_1130_1_WIRE_DUAL_128
NISWITCH_TOPOLOGY_1130_1_WIRE_4X64_MATRIX
NISWITCH_TOPOLOGY_1130_1_WIRE_8x32_MATRIX
NISWITCH_TOPOLOGY_1130_1_WIRE_OCTAL_32X1_MUX
NISWITCH_TOPOLOGY_1130_1_WIRE_QUAD_64X1_MUX
NISWITCH_TOPOLOGY_1130_1_WIRE_SIXTEEN_16X1_MUX
NISWITCH_TOPOLOGY_1130_2_WIRE_4X32_MATRIX
NISWITCH_TOPOLOGY_1130_2_WIRE_128X1_MUX NISWITCH_TOPOLOGY_1130_2_WIRE_OCTAL_16
NISWITCH_TOPOLOGY_1130_2_WIRE_QUAD_32X1_MUX

NISWITCH_TOPOLOGY_1130_4_WIRE_64X1_MUX NISWITCH_TOPOLOGY_1130_4_WIRE_QUAD_16X
NISWITCH_TOPOLOGY_1130_INDEPENDENT NISWITCH_TOPOLOGY_1160_16_SPDT
NISWITCH_TOPOLOGY_1161_8_SPDT NISWITCH_TOPOLOGY_1163R_OCTAL_4X1_MUX
NISWITCH_TOPOLOGY_1166_16_DPDT NISWITCH_TOPOLOGY_1166_32_SPDT
NISWITCH_TOPOLOGY_1167_INDEPENDENT NISWITCH_TOPOLOGY_1169_100_SPST
NISWITCH_TOPOLOGY_1169_50_DPST NISWITCH_TOPOLOGY_1175_1_WIRE_196X1_MUX
NISWITCH_TOPOLOGY_1175_2_WIRE_98X1_MUX NISWITCH_TOPOLOGY_1175_2_WIRE_95X1_MUX
NISWITCH_TOPOLOGY_1190_QUAD_4X1_MUX NISWITCH_TOPOLOGY_1191_QUAD_4X1_MUX
NISWITCH_TOPOLOGY_1192_8_SPDT NISWITCH_TOPOLOGY_1193_32X1_MUX
NISWITCH_TOPOLOGY_1193_16X1_TERMINATED_MUX
NISWITCH_TOPOLOGY_1193_DUAL_16X1_MUX NISWITCH_TOPOLOGY_1193_DUAL_8X1_TERMINAT
NISWITCH_TOPOLOGY_1193_QUAD_8X1_MUX NISWITCH_TOPOLOGY_1193_QUAD_4X1_TERMINAT
NISWITCH_TOPOLOGY_1193_INDEPENDENT NISWITCH_TOPOLOGY_1194_QUAD_4X1_MUX
NISWITCH_TOPOLOGY_1195_QUAD_4X1_MUX NISWITCH_TOPOLOGY_2501_1_WIRE_48X1_MUX
NISWITCH_TOPOLOGY_2501_1_WIRE_48X1_AMPLIFIED_MUX
NISWITCH_TOPOLOGY_2501_2_WIRE_24X1_MUX NISWITCH_TOPOLOGY_2501_2_WIRE_24X1_AMPL
NISWITCH_TOPOLOGY_2501_2_WIRE_DUAL_12X1_MUX
NISWITCH_TOPOLOGY_2501_2_WIRE_QUAD_6X1_MUX
NISWITCH_TOPOLOGY_2501_2_WIRE_4X6_MATRIX
NISWITCH_TOPOLOGY_2501_4_WIRE_12X1_MUX NISWITCH_TOPOLOGY_2503_1_WIRE_48X1_MUX
NISWITCH_TOPOLOGY_2503_2_WIRE_24X1_MUX NISWITCH_TOPOLOGY_2503_2_WIRE_DUAL_12X
NISWITCH_TOPOLOGY_2503_2_WIRE_QUAD_6X1_MUX
NISWITCH_TOPOLOGY_2503_2_WIRE_4X6_MATRIX
NISWITCH_TOPOLOGY_2503_4_WIRE_12X1_MUX NISWITCH_TOPOLOGY_2510_INDEPENDENT
NISWITCH_TOPOLOGY_2512_INDEPENDENT NISWITCH_TOPOLOGY_2514_INDEPENDENT
NISWITCH_TOPOLOGY_2515_INDEPENDENT NISWITCH_TOPOLOGY_2520_80_SPST
NISWITCH_TOPOLOGY_2521_40_DPST NISWITCH_TOPOLOGY_2522_53_SPDT
NISWITCH_TOPOLOGY_2523_26_DPDT NISWITCH_TOPOLOGY_2524_1_WIRE_128X1_MUX
NISWITCH_TOPOLOGY_2524_1_WIRE_DUAL_64X1_MUX
NISWITCH_TOPOLOGY_2524_1_WIRE_QUAD_32X1_MUX
NISWITCH_TOPOLOGY_2524_1_WIRE_OCTAL_16X1_MUX
NISWITCH_TOPOLOGY_2524_1_WIRE_SIXTEEN_8X1_MUX
NISWITCH_TOPOLOGY_2525_2_WIRE_64X1_MUX NISWITCH_TOPOLOGY_2525_2_WIRE_DUAL_32X
NISWITCH_TOPOLOGY_2525_2_WIRE_QUAD_16X1_MUX
NISWITCH_TOPOLOGY_2525_2_WIRE_OCTAL_8X1_MUX
NISWITCH_TOPOLOGY_2525_2_WIRE_SIXTEEN_4X1_MUX
NISWITCH_TOPOLOGY_2526_1_WIRE_158X1_MUX NISWITCH_TOPOLOGY_2526_2_WIRE_79X1_MU
NISWITCH_TOPOLOGY_2527_1_WIRE_64X1_MUX NISWITCH_TOPOLOGY_2527_1_WIRE_DUAL_32X
NISWITCH_TOPOLOGY_2527_2_WIRE_32X1_MUX NISWITCH_TOPOLOGY_2527_2_WIRE_DUAL_16X
NISWITCH_TOPOLOGY_2527_4_WIRE_16X1_MUX NISWITCH_TOPOLOGY_2527_INDEPENDENT
NISWITCH_TOPOLOGY_2529_2_WIRE_DUAL_4X16_MATRIX
NISWITCH_TOPOLOGY_2529_2_WIRE_8X16_MATRIX
NISWITCH_TOPOLOGY_2529_2_WIRE_4X32_MATRIX
NISWITCH_TOPOLOGY_2530_1_WIRE_128X1_MUX NISWITCH_TOPOLOGY_2530_1_WIRE_DUAL_64
NISWITCH_TOPOLOGY_2530_1_WIRE_4x32_MATRIX
NISWITCH_TOPOLOGY_2530_1_WIRE_8x16_MATRIX
NISWITCH_TOPOLOGY_2530_1_WIRE_OCTAL_16X1_MUX
NISWITCH_TOPOLOGY_2530_1_WIRE_QUAD_32X1_MUX
NISWITCH_TOPOLOGY_2530_2_WIRE_4x16_MATRIX
NISWITCH_TOPOLOGY_2530_2_WIRE_64X1_MUX NISWITCH_TOPOLOGY_2530_2_WIRE_DUAL_32X
NISWITCH_TOPOLOGY_2530_2_WIRE_QUAD_16X1_MUX
NISWITCH_TOPOLOGY_2530_4_WIRE_32X1_MUX NISWITCH_TOPOLOGY_2530_4_WIRE_DUAL_16X
NISWITCH_TOPOLOGY_2530_INDEPENDENT NISWITCH_TOPOLOGY_2531_1_WIRE_4X128_MATRIX
NISWITCH_TOPOLOGY_2531_1_WIRE_8X64_MATRIX

NISWITCH_TOPOLOGY_2531_1_WIRE_DUAL_4X64_MATRIX
NISWITCH_TOPOLOGY_2531_1_WIRE_DUAL_8X32_MATRIX
NISWITCH_TOPOLOGY_2531_2_WIRE_4X64_MATRIX NISWITCH_TOPOLOGY_2531_2_WIRE_8X32_M
NISWITCH_TOPOLOGY_2532_1_WIRE_16X32_MATRIX
NISWITCH_TOPOLOGY_2532_1_WIRE_4X128_MATRIX NISWITCH_TOPOLOGY_2532_1_WIRE_8X64_M
NISWITCH_TOPOLOGY_2532_1_WIRE_DUAL_16X16_MATRIX NISWITCH_TOPOLOGY_2532_1_WIRE_
NISWITCH_TOPOLOGY_2532_1_WIRE_DUAL_8X32_MATRIX NISWITCH_TOPOLOGY_2532_1_WIRE_S
NISWITCH_TOPOLOGY_2532_2_WIRE_16X16_MATRIX NISWITCH_TOPOLOGY_2532_2_WIRE_4X64_M
NISWITCH_TOPOLOGY_2532_2_WIRE_8X32_MATRIX NISWITCH_TOPOLOGY_2532_2_WIRE_DUAL_4
NISWITCH_TOPOLOGY_2533_1_WIRE_4X64_MATRIX NISWITCH_TOPOLOGY_2534_1_WIRE_8X32_M
NISWITCH_TOPOLOGY_2535_1_WIRE_4X136_MATRIX NISWITCH_TOPOLOGY_2536_1_WIRE_8X68_M
NISWITCH_TOPOLOGY_2540_1_WIRE_8X9_MATRIX NISWITCH_TOPOLOGY_2541_1_WIRE_8X12_MA
NISWITCH_TOPOLOGY_2542_QUAD_2X1_TERMINATED_MUX NISWITCH_TOPOLOGY_2543_DUAL_4
NISWITCH_TOPOLOGY_2544_8X1_TERMINATED_MUX NISWITCH_TOPOLOGY_2545_4X1_TERMINAT
NISWITCH_TOPOLOGY_2546_DUAL_4X1_MUX NISWITCH_TOPOLOGY_2547_8X1_MUX
NISWITCH_TOPOLOGY_2548_4_SPDT NISWITCH_TOPOLOGY_2549_TERMINATED_2_SPDT
NISWITCH_TOPOLOGY_2554_4X1_MUX NISWITCH_TOPOLOGY_2555_4X1_TERMINATED_MUX
NISWITCH_TOPOLOGY_2556_DUAL_4X1_MUX NISWITCH_TOPOLOGY_2557_8X1_MUX
NISWITCH_TOPOLOGY_2558_4_SPDT NISWITCH_TOPOLOGY_2559_TERMINATED_2_SPDT
NISWITCH_TOPOLOGY_2564_16_SPST NISWITCH_TOPOLOGY_2564_8_DPST
NISWITCH_TOPOLOGY_2565_16_SPST NISWITCH_TOPOLOGY_2566_16_SPDT
NISWITCH_TOPOLOGY_2566_8_DPDT NISWITCH_TOPOLOGY_2567_INDEPENDENT
NISWITCH_TOPOLOGY_2568_15_DPST NISWITCH_TOPOLOGY_2568_31_SPST
NISWITCH_TOPOLOGY_2569_100_SPST NISWITCH_TOPOLOGY_2569_50_DPST
NISWITCH_TOPOLOGY_2570_20_DPDT NISWITCH_TOPOLOGY_2570_40_SPDT
NISWITCH_TOPOLOGY_2571_66_SPDT NISWITCH_TOPOLOGY_2575_1_WIRE_196X1_MUX
NISWITCH_TOPOLOGY_2575_2_WIRE_98X1_MUX NISWITCH_TOPOLOGY_2575_2_WIRE_95X1_MUX
NISWITCH_TOPOLOGY_2576_2_WIRE_64X1_MUX NISWITCH_TOPOLOGY_2576_2_WIRE_DUAL_32X
NISWITCH_TOPOLOGY_2576_2_WIRE_OCTAL_8X1_MUX NISWITCH_TOPOLOGY_2576_2_WIRE_QUA
NISWITCH_TOPOLOGY_2576_2_WIRE_SIXTEEN_4X1_MUX NISWITCH_TOPOLOGY_2576_INDEPEND
NISWITCH_TOPOLOGY_2584_1_WIRE_12X1_MUX NISWITCH_TOPOLOGY_2584_1_WIRE_DUAL_6X1_
NISWITCH_TOPOLOGY_2584_2_WIRE_6X1_MUX NISWITCH_TOPOLOGY_2584_INDEPENDENT
NISWITCH_TOPOLOGY_2585_1_WIRE_10X1_MUX NISWITCH_TOPOLOGY_2586_10_SPST
NISWITCH_TOPOLOGY_2586_5_DPST NISWITCH_TOPOLOGY_2590_4X1_MUX
NISWITCH_TOPOLOGY_2591_4X1_MUX NISWITCH_TOPOLOGY_2593_16X1_MUX
NISWITCH_TOPOLOGY_2593_8X1_TERMINATED_MUX NISWITCH_TOPOLOGY_2593_DUAL_8X1_MU
NISWITCH_TOPOLOGY_2593_DUAL_4X1_TERMINATED_MUX NISWITCH_TOPOLOGY_2593_INDEPE
NISWITCH_TOPOLOGY_2594_4X1_MUX NISWITCH_TOPOLOGY_2595_4X1_MUX
NISWITCH_TOPOLOGY_2596_DUAL_6X1_MUX NISWITCH_TOPOLOGY_2597_6X1_TERMINATED_MU
NISWITCH_TOPOLOGY_2598_DUAL_TRANSFER NISWITCH_TOPOLOGY_2599_2_SPDT
NISWITCH_TOPOLOGY_2720_INDEPENDENT NISWITCH_TOPOLOGY_2722_INDEPENDENT
NISWITCH_TOPOLOGY_2725_INDEPENDENT NISWITCH_TOPOLOGY_2727_INDEPENDENT
NISWITCH_TOPOLOGY_2737_2_WIRE_4X64_MATRIX NISWITCH_TOPOLOGY_2738_2_WIRE_8X32_M
NISWITCH_TOPOLOGY_2739_2_WIRE_16X16_MATRIX NISWITCH_TOPOLOGY_2746_QUAD_4X1_MU
NISWITCH_TOPOLOGY_2747_DUAL_8X1_MUX NISWITCH_TOPOLOGY_2748_16X1_MUX
NISWITCH_TOPOLOGY_2790_INDEPENDENT NISWITCH_TOPOLOGY_2796_DUAL_6X1_MUX
NISWITCH_TOPOLOGY_2797_6X1_TERMINATED_MUX NISWITCH_TOPOLOGY_2798_DUAL_TRANSF
NISWITCH_TOPOLOGY_2799_2_SPDT

- **`simulate`** (*`bool`*) – Enables simulation of the switch module specified in the resource name parameter. Valid Values: True - simulate False - Don't simulate (Default Value)
- **`reset_device`** (*`bool`*) – Specifies whether to reset the switch module during the initialization process. Valid Values: True - Reset Device (Default Value) False - Currently unsupported. The device will not reset.

Methods

abort

`niswitch.Session.abort()`

Aborts the scan in progress. Initiate a scan with `niswitch.Session.initiate()`. If the switch module is not scanning, NISWITCH_ERROR_NO_SCAN_IN_PROGRESS error is returned.

can_connect

`niswitch.Session.can_connect(channel1, channel2)`

Verifies that a path between channel 1 and channel 2 can be created. If a path is possible in the switch module, the availability of that path is returned given the existing connections. If the path is possible but in use, a NISWITCH_WARN_IMPLICIT_CONNECTION_EXISTS warning is returned.

Parameters

- **channel1** (*str*) – Input one of the channel names of the desired path. Pass the other channel name as the channel 2 parameter. Refer to Devices Overview for valid channel names for the switch module. Examples of valid channel names: ch0, com0, ab0, r1, c2, cjtemp Default value: “”
- **channel2** (*str*) – Input one of the channel names of the desired path. Pass the other channel name as the channel 1 parameter. Refer to Devices Overview for valid channel names for the switch module. Examples of valid channel names: ch0, com0, ab0, r1, c2, cjtemp Default value: “”

Return type `niswitch.PathCapability`

Returns

Indicates whether a path is valid. Possible values include:

- `PATH_AVAILABLE` 1
- `PATH_EXISTS` 2
- `PATH_UNSUPPORTED` 3
- `RESOURCE_IN_USE` 4
- `SOURCE_CONFLICT` 5
- `CHANNEL_NOT_AVAILABLE` 6

Notes: (1) `PATH_AVAILABLE` indicates that the driver can create the path at this time. (2) `PATH_EXISTS` indicates that the path already exists. (3) `PATH_UNSUPPORTED` indicates that the instrument is not capable of creating a path between the channels you specify. (4) `RESOURCE_IN_USE` indicates that although the path is valid, the driver cannot create the path at this moment because the switch device is currently using one or more of the required channels to create another path. You must destroy the other path before creating this one. (5) `SOURCE_CONFLICT` indicates that the instrument cannot create a path because both channels are connected to a different source channel. (6) `CHANNEL_NOT_AVAILABLE` indicates that the driver cannot create a path between the two channels because one of the channels is a configuration channel and thus unavailable for external connections.

close

`niswitch.Session.close()`

Terminates the NI-SWITCH session and all of its properties and deallocates any memory resources the driver uses. Notes: (1) You must unlock the session before calling `niswitch.Session._close()`. (2) After calling `niswitch.Session._close()`, you cannot use the instrument driver again until you call `niswitch.Session.init()` or `niswitch.Session.InitWithOptions()`.

Note: One or more of the referenced methods are not in the Python API for this driver.

Note: This method is not needed when using the session context manager

commit

`niswitch.Session.commit()`

Downloads the configured scan list and trigger settings to hardware. Calling *`niswitch.Session.commit()`* optional as it is implicitly called during *`niswitch.Session.initiate()`*. Use *`niswitch.Session.commit()`* to arm triggers in a given order or to control when expensive hardware operations are performed.

connect

`niswitch.Session.connect(`*channel1*, *channel2*`)`

Creates a path between channel 1 and channel 2. The driver calculates and uses the shortest path between the two channels. Refer to Immediate Operations for information about Channel Usage types. If a path is not available, the method returns one of the following errors: - NISWITCH_ERROR_EXPLICIT_CONNECTION_EXISTS, if the two channels are already explicitly connected by calling either the *`niswitch.Session.connect()`* or *`niswitch.Session.set_path()`* method. - NISWITCH_ERROR_IS_CONFIGURATION_CHANNEL, if a channel is a configuration channel. Error elaboration contains information about which of the two channels is a configuration channel. - NISWITCH_ERROR_ATTEMPT_TO_CONNECT_SOURCES, if both channels are connected to a different source. Error elaboration contains information about sources channel 1 and 2 connect to. - NISWITCH_ERROR_CANNOT_CONNECT_TO_ITSELF, if channels 1 and 2 are one and the same channel. - NISWITCH_ERROR_PATH_NOT_FOUND, if the driver cannot find a path between the two channels. Note: Paths are bidirectional. For example, if a path exists between channels CH1 and CH2, then the path also exists between channels CH2 and CH1.

Parameters

- **`channel1`** (*`str`*) – Input one of the channel names of the desired path. Pass the other channel name as the channel 2 parameter. Refer to Devices Overview for valid channel names for the switch module. Examples of valid channel names: ch0, com0, ab0, r1, c2, cjtemp Default value: None
- **`channel2`** (*`str`*) – Input one of the channel names of the desired path. Pass the other channel name as the channel 1 parameter. Refer to Devices Overview for valid channel names for the switch module. Examples of valid channel names: ch0, com0, ab0, r1, c2, cjtemp Default value: None

connect_multiple

niswitch.Session.**connect_multiple**(*connection_list*)

Creates the connections between channels specified in Connection List. Specify connections with two endpoints only or the explicit path between two endpoints. NI-SWITCH calculates and uses the shortest path between the channels. Refer to Setting Source and Configuration Channels for information about channel usage types. In the event of an error, connecting stops at the point in the list where the error occurred. If a path is not available, the method returns one of the following errors: - NISWITCH_ERROR_EXPLICIT_CONNECTION_EXISTS, if the two channels are already explicitly connected. - NISWITCH_ERROR_IS_CONFIGURATION_CHANNEL, if a channel is a configuration channel. Error elaboration contains information about which of the two channels is a configuration channel. - NISWITCH_ERROR_ATTEMPT_TO_CONNECT_SOURCES, if both channels are connected to a different source. Error elaboration contains information about sources channel 1 and 2 to connect. - NISWITCH_ERROR_CANNOT_CONNECT_TO_ITSELF, if channels 1 and 2 are one and the same channel. - NISWITCH_ERROR_PATH_NOT_FOUND, if the driver cannot find a path between the two channels. Note: Paths are bidirectional. For example, if a path exists between channels ch1 and ch2, then the path also exists between channels ch1 and ch2.

Parameters **connection_list** (*str*) – Connection List specifies a list of connections between channels to make. NI-SWITCH validates the connection list, and aborts execution of the list if errors are returned. Refer to Connection and Disconnection List Syntax for valid connection list syntax and examples. Refer to Devices Overview for valid channel names for the switch module. Example of a valid connection list: c0 -> r1, [c2 -> r2 -> c3] In this example, r2 is a configuration channel. Default value: None

disable

niswitch.Session.**disable**()

Places the switch module in a quiescent state where it has minimal or no impact on the system to which it is connected. All channels are disconnected and any scan in progress is aborted.

disconnect

niswitch.Session.**disconnect**(*channel1*, *channel2*)

This method destroys the path between two channels that you create with the `niswitch.Session.connect()` or `niswitch.Session.set_path()` method. If a path is not connected or not available, the method returns the IVISWTCH_ERROR_NO_SUCH_PATH error.

Parameters

- **channel1** (*str*) – Input one of the channel names of the path to break. Pass the other channel name as the channel 2 parameter. Refer to Devices Overview for valid channel names for the switch module. Examples of valid channel names: ch0, com0, ab0, r1, c2, cjtemp Default value: None
- **channel2** (*str*) – Input one of the channel names of the path to break. Pass the other channel name as the channel 1 parameter. Refer to Devices Overview for valid channel names for the switch module. Examples of valid channel names: ch0, com0, ab0, r1, c2, cjtemp Default value: None

disconnect_all

niswitch.Session.**disconnect_all**()

Breaks all existing paths. If the switch module cannot break all paths, NISWITCH_WARN_PATH_REMAINS warning is returned.

disconnect_multiple

niswitch.Session.**disconnect_multiple**(*disconnection_list*)

Breaks the connections between channels specified in Disconnection List. If no connections exist between channels, NI-SWITCH returns an error. In the event of an error, the VI stops at the point in the list where the error occurred.

Parameters **disconnection_list** (*str*) – Disconnection List specifies a list of connections between channels to break. NI-SWITCH validates the disconnection list, and aborts execution of the list if errors are returned. Refer to Connection and Disconnection List Syntax for valid disconnection list syntax and examples. Refer to Devices Overview for valid channel names for the switch module. Example of a valid disconnection list: c0 -> r1, [c2 -> r2 -> c3] In this example, r2 is a configuration channel. Default value: None

get_channel_name

niswitch.Session.**get_channel_name**(*index*)

Returns the channel string that is in the channel table at the specified index. Use *niswitch.Session.get_channel_name()* in a For Loop to get a complete list of valid channel names for the switch module. Use the Channel Count property to determine the number of channels.

Parameters **index** (*int*) – A 1-based index into the channel table. Default value: 1 Maximum value: Value of Channel Count property.

Return type str

Returns Returns the channel name that is in the channel table at the index you specify.

get_path

niswitch.Session.**get_path**(*channel1*, *channel2*)

Returns a string that identifies the explicit path created with *niswitch.Session.connect()*. Pass this string to *niswitch.Session.set_path()* to establish the exact same path in future connections. In some cases, multiple paths are available between two channels. When you call *niswitch.Session.connect()*, the driver selects an available path. With *niswitch.Session.connect()*, there is no guarantee that the driver selected path will always be the same path through the switch module. *niswitch.Session.get_path()* only returns those paths explicitly created by niSwitch Connect Channels or *niswitch.Session.set_path()*. For example, if you connect channels CH1 and CH3,and then channels CH2 and CH3, an explicit path between channels CH1 and CH2 does not exist an error is returned

Parameters

- **channel1** (*str*) – Input one of the channel names of the desired path. Pass the other channel name as the channel 2 parameter. Refer to Devices Overview for valid channel names for the switch module. Examples of valid channel names: ch0, com0, ab0, r1, c2, cjtemp Default value: “”

- **channel2** (*str*) – Input one of the channel names of the desired path. Pass the other channel name as the channel 1 parameter. Refer to Devices Overview for valid channel names for the switch module. Examples of valid channel names: ch0, com0, ab0, r1, c2, cjtemp Default value: “”

Return type str

Returns A string composed of comma-separated paths between channel 1 and channel 2. The first and last names in the path are the endpoints of the path. All other channels in the path are configuration channels. Examples of returned paths: ch0->com0, com0->ab0

get_relay_count

`niswitch.Session.`**`get_relay_count`**`(`*relay_name*`)`

Returns the number of times the relay has changed from Closed to Open. Relay count is useful for tracking relay lifetime and usage. Call `niswitch.Session.wait_for_debounce()` before `niswitch.Session.get_relay_count()` to ensure an accurate count. Refer to the Relay Count topic in the NI Switches Help to determine if the switch module supports relay counting.

Parameters **relay_name** (*str*) – Name of the relay. Default value: None Examples of valid relay names: ch0, ab0, 1wire, hlselect Refer to Devices Overview for a list of valid relay names for the switch module.

Return type int

Returns The number of relay cycles.

get_relay_name

`niswitch.Session.`**`get_relay_name`**`(`*index*`)`

Returns the relay name string that is in the relay list at the specified index. Use `niswitch.Session.get_relay_name()` in a For Loop to get a complete list of valid relay names for the switch module. Use the Number of Relays property to determine the number of relays.

Parameters **index** (*int*) – A 1-based index into the channel table. Default value: 1 Maximum value: Value of Channel Count property.

Return type str

Returns Returns the relay name for the index you specify.

get_relay_position

`niswitch.Session.`**`get_relay_position`**`(`*relay_name*`)`

Returns the relay position for the relay specified in the Relay Name parameter.

Parameters **relay_name** (*str*) – Name of the relay. Default value: None Examples of valid relay names: ch0, ab0, 1wire, hlselect Refer to Devices Overview for a list of valid relay names for the switch module.

Return type `niswitch.RelayPosition`

Returns Indicates whether the relay is open or closed. `OPEN` 10 `CLOSED` 11

initiate

`niswitch.Session.initiate()`

Commits the configured scan list and trigger settings to hardware and initiates the scan. If niSwitch Commit was called earlier, niSwitch Initiate Scan only initiates the scan and returns immediately. Once the scanning operation begins, you cannot perform any other operation other than GetAttribute, AbortScan, or SendSoftwareTrigger. All other methods return NISWITCH_ERROR_SCAN_IN_PROGRESS. To stop the scanning operation, To stop the scanning operation, call *niswitch.Session.abort()*.

Note: This method will return a Python context manager that will initiate on entering and abort on exit.

lock

`niswitch.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 *niswitch.Session.lock()* method.
- A call to NI-SWITCH locked the session.
- After a call to the *niswitch.Session.lock()* method returns successfully, no other threads can access the device session until you call the *niswitch.Session.unlock()* method or exit out of the with block when using lock context manager.
- Use the *niswitch.Session.lock()* method and the *niswitch.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 *niswitch.Session.lock()* method within the same thread. To completely unlock the session, you must balance each call to the *niswitch.Session.lock()* method with a call to the *niswitch.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

```python
with niswitch.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, *niswitch.Session.lock()* acts as a context manager and unlock will be called when the *with* block is exited

relay_control

niswitch.Session.**relay_control**(*relay_name*, *relay_action*)

Controls individual relays of the switch. When controlling individual relays, the protection offered by setting the usage of source channels and configuration channels, and by enabling or disabling analog bus sharing on the NI SwitchBlock, does not apply. Refer to the device book for your switch in the NI Switches Help to determine if the switch supports individual relay control.

Parameters

- **relay_name** (*str*) – Name of the relay. Default value: None Examples of valid relay names: ch0, ab0, 1wire, hlselect Refer to Devices Overview for a list of valid relay names for the switch module.
- **relay_action** (*niswitch.RelayAction*) – Specifies whether to open or close a given relay. Default value: Relay Close Defined values: *OPEN* *CLOSE* (Default Value)

reset

niswitch.Session.**reset**()

Disconnects all created paths and returns the switch module to the state at initialization. Configuration channel and source channel settings remain unchanged.

reset_with_defaults

niswitch.Session.**reset_with_defaults**()

Resets the switch module and applies initial user specified settings from the logical name used to initialize the session. If the session was created without a logical name, this method is equivalent to *niswitch.Session.reset()*.

route_scan_advanced_output

niswitch.Session.**route_scan_advanced_output**(*scan_advanced_output_connector*, *scan_advanced_output_bus_line*, *invert=False*)

Routes the scan advanced output trigger from a trigger bus line (TTLx) to the front or rear connector.

Parameters

- **scan_advanced_output_connector** (*niswitch.ScanAdvancedOutput*) – The scan advanced trigger destination. Valid locations are the *FRONTCONNECTOR* and *REARCONNECTOR*. Default value: *FRONTCONNECTOR*

  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.

- **scan_advanced_output_bus_line** (*niswitch.ScanAdvancedOutput*) – The trigger line to route the scan advanced output trigger from the front or rear connector. Select *NONE* to break an existing route. Default value: None Valid Values: *NONE TTL0 TTL1 TTL2 TTL3 TTL4 TTL5 TTL6 TTL7*

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.

- **invert** (*bool*) – If True, inverts the input trigger signal from falling to rising or vice versa. Default value: False

route_trigger_input

`niswitch.Session.`**`route_trigger_input`**(*trigger_input_connector*, *trigger_input_bus_line*, *invert=False*)

Routes the input trigger from the front or rear connector to a trigger bus line (TTLx). To disconnect the route, call this method again and specify None for trigger bus line parameter.

Parameters

- **trigger_input_connector** (*niswitch.TriggerInput*) – The location of the input trigger source on the switch module. Valid locations are the *FRONTCONNECTOR* and *REARCONNECTOR*. Default value: *FRONTCONNECTOR*

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.

- **trigger_input_bus_line** (*niswitch.TriggerInput*) – The trigger line to route the input trigger. Select `NISWITCH_VAL_NONE` to break an existing route. Default value: None Valid Values: `NISWITCH_VAL_NONE` *TTL0 TTL1 TTL2 TTL3 TTL4 TTL5 TTL6 TTL7*

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.

- **invert** (*bool*) – If True, inverts the input trigger signal from falling to rising or vice versa. Default value: False

self_test

`niswitch.Session.`**`self_test`**()

Verifies that the driver can communicate with the switch module.

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

`niswitch.Session.`**`send_software_trigger`**`()`

Sends a software trigger to the switch module specified in the NI-SWITCH session. When the trigger input is set to *`SOFTWARE_TRIG`* through either the `niswitch.Session.ConfigureScanTrigger()` or the *`niswitch.Session.trigger_input`* property, the scan does not proceed from a semi-colon (wait for trigger) until *`niswitch.Session.send_software_trigger()`* is called.

Note: One or more of the referenced methods are not in the Python API for this driver.

set_path

`niswitch.Session.`**`set_path`**`(`*path_list*`)`

Connects two channels by specifying an explicit path in the path list parameter. *`niswitch.Session.set_path()`* is particularly useful where path repeatability is important, such as in calibrated signal paths. If this is not necessary, use *`niswitch.Session.connect()`*.

Parameters **`path_list`** (*`str`*) – A string composed of comma-separated paths between channel 1 and channel 2. The first and last names in the path are the endpoints of the path. Every other channel in the path are configuration channels. Example of a valid path list string: ch0->com0, com0->ab0. In this example, com0 is a configuration channel. Default value: None Obtain the path list for a previously created path with *`niswitch.Session.get_path()`*.

unlock

`niswitch.Session.`**`unlock`**`()`

Releases a lock that you acquired on an device session using *`niswitch.Session.lock()`*. Refer to *`niswitch.Session.unlock()`* for additional information on session locks.

wait_for_debounce

`niswitch.Session.`**`wait_for_debounce`**`(`*maximum_time_ms=hightime.timedelta(milliseconds=5000)*`)`

Pauses until all created paths have settled. If the time you specify with the Maximum Time (ms) parameter elapsed before the switch paths have settled, this method returns the NISWITCH_ERROR_MAX_TIME_EXCEEDED error.

Parameters **`maximum_time_ms`** (*`hightime.timedelta, datetime.timedelta, or int in milliseconds`*) – Specifies the maximum length of time to wait for all relays in the switch module to activate or deactivate. If the specified time elapses before all relays active or deactivate, a timeout error is returned. Default Value:5000 ms

wait_for_scan_complete

`niswitch.Session.`**`wait_for_scan_complete`**`(`*maximum_time_ms=hightime.timedelta(milliseconds=5000)*`)`

Pauses until the switch module stops scanning or the maximum time has elapsed and returns a timeout error. If the time you specify with the Maximum Time (ms) parameter elapsed before the scanning operation has finished, this method returns the NISWITCH_ERROR_MAX_TIME_EXCEEDED error.

Parameters **`maximum_time_ms`** (*`hightime.timedelta`*, *`datetime.timedelta`*, *`or int in milliseconds`*) – Specifies the maximum length of time to wait for the switch module to stop scanning. If the specified time elapses before the scan ends, NISWITCH_ERROR_MAX_TIME_EXCEEDED error is returned. Default Value:5000 ms

Properties

analog_bus_sharing_enable

`niswitch.Session.`**`analog_bus_sharing_enable`**

Enables or disables sharing of an analog bus line so that multiple NI SwitchBlock devices may connect to it simultaneously. To enable multiple NI SwitchBlock devices to share an analog bus line, set this property to True for each device on the channel that corresponds with the shared analog bus line. The default value for all devices is False, which disables sharing of the analog bus. Refer to the Using the Analog Bus on an NI SwitchBlock Carrier topic in the NI Switches Help for more information about sharing the analog bus.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].analog_bus_sharing_enable`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.analog_bus_sharing_enable`

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: **Channel Configuration:Analog Bus Sharing Enable**
- C Attribute: **NISWITCH_ATTR_ANALOG_BUS_SHARING_ENABLE**

bandwidth

`niswitch.Session.`**`bandwidth`**

This channel-based property returns the bandwidth for the channel. The units are hertz.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].bandwidth`

To set/get on all channels, you can call the property directly on the `niswitch.Session`.

Example: `my_session.bandwidth`

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:

- LabVIEW Property: **Module Characteristics:Bandwidth**
- C Attribute: **NISWITCH_ATTR_BANDWIDTH**

channel_count

`niswitch.Session.`**`channel_count`**

Indicates the number of channels that the specific instrument driver supports.

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: **NISWITCH_ATTR_CHANNEL_COUNT**

characteristic_impedance

`niswitch.Session.`**`characteristic_impedance`**

This channel-based property returns the characteristic impedance for the channel. The units are ohms.

Tip: This property can be set/get on specific channels within your `niswitch.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].characteristic_impedance`

To set/get on all channels, you can call the property directly on the `niswitch.Session`.

Example: `my_session.characteristic_impedance`

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:

- LabVIEW Property: **Module Characteristics:Characteristic Impedance**
- C Attribute: **NISWITCH_ATTR_CHARACTERISTIC_IMPEDANCE**

continuous_scan

`niswitch.Session.`**`continuous_scan`**

When a switch device is scanning, the swich can either stop scanning when the end of the scan (False) or continue scanning from the top of the scan list again (True). Notice that if you set the scan to continuous (True), the Wait For Scan Complete operation will always time out and you must call Abort to stop the scan.

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: **Scanning Configuration:Continuous Scan**
- C Attribute: **NISWITCH_ATTR_CONTINUOUS_SCAN**

digital_filter_enable

`niswitch.Session.`**`digital_filter_enable`**

This property specifies whether to apply the pulse width filter to the Trigger Input. Enabling the Digital Filter (True) prevents the switch module from being triggered by pulses that are less than 150 ns on PXI trigger lines 0–7. When Digital Filter is disabled (False), it is possible for the switch module to be triggered by noise on the PXI trigger lines. If the device triggering the switch is capable of sending pulses greater than 150 ns, you should not disable the Digital Filter.

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: **Scanning Configuration:Digital Filter Enable**
- C Attribute: **NISWITCH_ATTR_DIGITAL_FILTER_ENABLE**

driver_setup

`niswitch.Session.`**`driver_setup`**

This property indicates the Driver Setup string that the user specified when initializing the driver. Some cases exist where the end-user must specify instrument driver options at initialization time. An example of this is specifying a particular instrument model from among a family of instruments that the driver supports. This is useful when using simulation. The end-user can specify driver-specific options through the DriverSetup keyword in the optionsString parameter to the `niswitch.Session.InitWithOptions()` method, or through the IVI Configuration Utility. If the user does not specify a Driver Setup string, this property returns an empty string.

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:Driver Setup**
- C Attribute: **NISWITCH_ATTR_DRIVER_SETUP**

handshaking_initiation

`niswitch.Session.`**`handshaking_initiation`**

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.HandshakingInitiation
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Scanning Configuration:Handshaking Initiation**
- C Attribute: **NISWITCH_ATTR_HANDSHAKING_INITIATION**

instrument_firmware_revision

`niswitch.Session.`**`instrument_firmware_revision`**

A string that contains the firmware revision information for the instrument you are currently using.

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:Firmware Revision**
- C Attribute: **NISWITCH_ATTR_INSTRUMENT_FIRMWARE_REVISION**

instrument_manufacturer

`niswitch.Session.`**`instrument_manufacturer`**

A string that contains the name of the instrument manufacturer you are currently using.

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: **NISWITCH_ATTR_INSTRUMENT_MANUFACTURER**

instrument_model

`niswitch.Session.`**`instrument_model`**

A string that contains the model number or name of the instrument that you are currently using.

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: **NISWITCH_ATTR_INSTRUMENT_MODEL**

io_resource_descriptor

niswitch.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.

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:IO Resource Descriptor**
- C Attribute: **NISWITCH_ATTR_IO_RESOURCE_DESCRIPTOR**

is_configuration_channel

niswitch.Session.**is_configuration_channel**

This channel-based property specifies whether to reserve the channel for internal path creation. A channel that is available for internal path creation is called a configuration channel. The driver may use configuration channels to create paths between two channels you specify in the *niswitch.Session.connect()* method. Configuration channels are not available for external connections. Set this property to True to mark the channel as a configuration channel. Set this property to False to mark the channel as available for external connections. After you identify a channel as a configuration channel, you cannot use that channel for external connections. The *niswitch.Session.connect()* method returns the NISWITCH_ERROR_IS_CONFIGURATION_CHANNEL error when you attempt to establish a connection between a configuration channel and any other channel.

Tip: This property can be set/get on specific channels within your *niswitch.Session* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].is_configuration_channel`

To set/get on all channels, you can call the property directly on the *niswitch.Session*.

Example: `my_session.is_configuration_channel`

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: **Channel Configuration:Is Configuration Channel**
- C Attribute: **NISWITCH_ATTR_IS_CONFIGURATION_CHANNEL**

is_debounced

`niswitch.Session.`**`is_debounced`**

This property indicates whether the entire switch device has settled since the last switching command. A value of True indicates that all signals going through the switch device are valid.

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: **Module Characteristics:Is Debounced**
- C Attribute: **NISWITCH_ATTR_IS_DEBOUNCED**

is_scanning

`niswitch.Session.`**`is_scanning`**

If True, the switch module is currently scanning through the scan list (i.e. it is not in the Idle state). If False, the switch module is not currently scanning through the scan list (i.e. it is in the Idle state).

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: **Scanning Configuration:Is Scanning**
- C Attribute: **NISWITCH_ATTR_IS_SCANNING**

---

is_source_channel

`niswitch.Session.`**`is_source_channel`**

This channel-based property specifies whether you want to identify the channel as a source channel. Typically, you set this property to True when you attach the channel to a power supply, a method generator, or an active measurement point on the unit under test, and you do not want to connect the channel to another source. The driver prevents source channels from connecting to each other. The *`niswitch.Session.connect()`* method returns the NISWITCH_ERROR_ATTEMPT_TO_CONNECT_SOURCES when you attempt to connect two channels that you identify as source channels.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].is_source_channel`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.is_source_channel`

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: **Channel Configuration:Is Source Channel**
- C Attribute: **NISWITCH_ATTR_IS_SOURCE_CHANNEL**

---

is_waiting_for_trig

`niswitch.Session.`**`is_waiting_for_trig`**

In a scan list, a semi-colon (;) is used to indicate that at that point in the scan list, the scan engine should pause until a trigger is received from the trigger input. If that trigger is user generated through either a hardware pulse or the Send SW Trigger operation, it is necessary for the user to know when the scan engine has reached such a state.

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: **Scanning Configuration:Is Waiting for Trigger?**
- C Attribute: **NISWITCH_ATTR_IS_WAITING_FOR_TRIG**

logical_name

`niswitch.Session.`**`logical_name`**

A string containing the logical name you specified when opening the current IVI session. You may pass a logical name to the `niswitch.Session.init()` or `niswitch.Session.InitWithOptions()` methods. 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: **NISWITCH_ATTR_LOGICAL_NAME**

max_ac_voltage

`niswitch.Session.`**`max_ac_voltage`**

This channel-based property returns the maximum AC voltage the channel can switch. The units are volts RMS.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_ac_voltage`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.max_ac_voltage`

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:

- LabVIEW Property: **Module Characteristics:Maximum AC Voltage**
- C Attribute: **NISWITCH_ATTR_MAX_AC_VOLTAGE**

max_carry_ac_current

`niswitch.Session.`**`max_carry_ac_current`**

This channel-based property returns the maximum AC current the channel can carry. The units are amperes RMS.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_carry_ac_current`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.max_carry_ac_current`

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:

- LabVIEW Property: **Module Characteristics:Maximum Carry AC Current**
- C Attribute: **NISWITCH_ATTR_MAX_CARRY_AC_CURRENT**

max_carry_ac_power

`niswitch.Session.`**`max_carry_ac_power`**

This channel-based property returns the maximum AC power the channel can carry. The units are volt-amperes.

Tip: This property can be set/get on specific channels within your `niswitch.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_carry_ac_power`

To set/get on all channels, you can call the property directly on the `niswitch.Session`.

Example: `my_session.max_carry_ac_power`

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:

- LabVIEW Property: **Module Characteristics:Maximum Carry AC Power**
- C Attribute: **NISWITCH_ATTR_MAX_CARRY_AC_POWER**

max_carry_dc_current

`niswitch.Session.`**`max_carry_dc_current`**

This channel-based property returns the maximum DC current the channel can carry. The units are amperes.

Tip: This property can be set/get on specific channels within your `niswitch.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_carry_dc_current`

To set/get on all channels, you can call the property directly on the `niswitch.Session`.

Example: `my_session.max_carry_dc_current`

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:

- LabVIEW Property: **Module Characteristics:Maximum Carry DC Current**
- C Attribute: **NISWITCH_ATTR_MAX_CARRY_DC_CURRENT**

max_carry_dc_power

`niswitch.Session.`**`max_carry_dc_power`**

This channel-based property returns the maximum DC power the channel can carry. The units are watts.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_carry_dc_power`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.max_carry_dc_power`

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:

- LabVIEW Property: **Module Characteristics:Maximum Carry DC Power**
- C Attribute: **NISWITCH_ATTR_MAX_CARRY_DC_POWER**

max_dc_voltage

`niswitch.Session.`**`max_dc_voltage`**

This channel-based property returns the maximum DC voltage the channel can switch. The units are volts.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_dc_voltage`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.max_dc_voltage`

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:

- LabVIEW Property: **Module Characteristics:Maximum DC Voltage**
- C Attribute: **NISWITCH_ATTR_MAX_DC_VOLTAGE**

max_switching_ac_current

`niswitch.Session.`**`max_switching_ac_current`**

This channel-based property returns the maximum AC current the channel can switch. The units are amperes RMS.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_switching_ac_current`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.max_switching_ac_current`

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:

- LabVIEW Property: **Module Characteristics:Maximum Switching AC Current**
- C Attribute: **NISWITCH_ATTR_MAX_SWITCHING_AC_CURRENT**

max_switching_ac_power

`niswitch.Session.`**`max_switching_ac_power`**

This channel-based property returns the maximum AC power the channel can switch. The units are volt-amperes.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_switching_ac_power`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.max_switching_ac_power`

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:

- LabVIEW Property: **Module Characteristics:Maximum Switching AC Power**
- C Attribute: **NISWITCH_ATTR_MAX_SWITCHING_AC_POWER**

max_switching_dc_current

`niswitch.Session.`**`max_switching_dc_current`**

This channel-based property returns the maximum DC current the channel can switch. The units are amperes.

Tip: This property can be set/get on specific channels within your `niswitch.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_switching_dc_current`

To set/get on all channels, you can call the property directly on the `niswitch.Session`.

Example: `my_session.max_switching_dc_current`

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:

- LabVIEW Property: **Module Characteristics:Maximum Switching DC Current**
- C Attribute: **NISWITCH_ATTR_MAX_SWITCHING_DC_CURRENT**

max_switching_dc_power

`niswitch.Session.`**`max_switching_dc_power`**

This channel-based property returns the maximum DC power the channel can switch. The units are watts.

Tip: This property can be set/get on specific channels within your `niswitch.Session` instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].max_switching_dc_power`

To set/get on all channels, you can call the property directly on the `niswitch.Session`.

Example: `my_session.max_switching_dc_power`

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:

- LabVIEW Property: **Module Characteristics:Maximum Switching DC Power**
- C Attribute: **NISWITCH_ATTR_MAX_SWITCHING_DC_POWER**

number_of_relays

`niswitch.Session.`**`number_of_relays`**

This property returns the number of relays.

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: **Module Characteristics:Number of Relays**
- C Attribute: **NISWITCH_ATTR_NUMBER_OF_RELAYS**

num_of_columns

`niswitch.Session.`**`num_of_columns`**

This property returns the number of channels on the column of a matrix or scanner. If the switch device is a scanner, this value is the number of input channels. The `niswitch.Session.wire_mode` property affects the number of available columns. For example, if your device has 8 input lines and you use the four-wire mode, then the number of columns you have available is 2.

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: **Matrix Configuration:Number of Columns**
- C Attribute: **NISWITCH_ATTR_NUM_OF_COLUMNS**

num_of_rows

`niswitch.Session.`**`num_of_rows`**

This property returns the number of channels on the row of a matrix or scanner. If the switch device is a scanner, this value is the number of output channels. The *`niswitch.Session.wire_mode`* property affects the number of available rows. For example, if your device has 8 input lines and you use the two-wire mode, then the number of columns you have available is 4.

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: **Matrix Configuration:Number of Rows**
- C Attribute: **NISWITCH_ATTR_NUM_OF_ROWS**

power_down_latching_relays_after_debounce

`niswitch.Session.`**`power_down_latching_relays_after_debounce`**

This property specifies whether to power down latching relays after calling Wait For Debounce. When Power Down Latching Relays After Debounce is enabled (True), a call to Wait For Debounce ensures that the relays are settled and the latching relays are powered down.

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: **Module Characteristics:Power Down Latching Relays After Debounce**
- C Attribute: **NISWITCH_ATTR_POWER_DOWN_LATCHING_RELAYS_AFTER_DEBOUNCE**

scan_advanced_output

`niswitch.Session.scan_advanced_output`

This property specifies the method you want to use to notify another instrument that all signals going through the switch device have settled following the processing of one entry in the scan list.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ScanAdvancedOutput
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Scanning Configuration:Scan Advanced Output**
- C Attribute: **NISWITCH_ATTR_SCAN_ADVANCED_OUTPUT**

scan_advanced_polarity

`niswitch.Session.scan_advanced_polarity`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.ScanAdvancedPolarity
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Scanning Configuration:Scan Advanced Polarity**
- C Attribute: **NISWITCH_ATTR_SCAN_ADVANCED_POLARITY**

scan_delay

`niswitch.Session.scan_delay`

This property specifies the minimum amount of time the switch device waits before it asserts the scan advanced output trigger after opening or closing the switch. The switch device always waits for debounce before asserting the trigger. The units are seconds. the greater value of the settling time and the value you specify as the scan delay.

Note: NI PXI-2501/2503/2565/2590/2591 Users–the actual delay will always be

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: **Scanning Configuration:Scan Delay**
- C Attribute: **NISWITCH_ATTR_SCAN_DELAY**

scan_list

`niswitch.Session.`**`scan_list`**

This property contains a scan list, which is a string that specifies channel connections and trigger conditions. The *`niswitch.Session.initiate()`* method makes or breaks connections and waits for triggers according to the instructions in the scan list. The scan list is comprised of channel names that you separate with special characters. These special characters determine the operations the scanner performs on the channels when it executes this scan list. To create a path between two channels, use the following character between the two channel names: -> (a dash followed by a '>' sign) Example: 'CH1->CH2' tells the switch to make a path from channel CH1 to channel CH2. To break or clear a path, use the following character as a prefix before the path: ~ (tilde) Example: '~CH1->CH2' tells the switch to break the path from channel CH1 to channel CH2. To tell the switch device to wait for a trigger event, use the following character as a separator between paths: ; (semi-colon) Example: 'CH1->CH2;CH3->CH4' tells the switch to make the path from channel CH1 to channel CH2, wait for a trigger, and then make the path from CH3 to CH4.

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: **Scanning Configuration:Scan List**
- C Attribute: **NISWITCH_ATTR_SCAN_LIST**

scan_mode

`niswitch.Session.`**`scan_mode`**

This property specifies what happens to existing connections that conflict with the connections you make in a scan list. For example, if CH1 is already connected to CH2 and the scan list instructs the switch device to connect CH1 to CH3, this property specifies what happens to the connection between CH1 and CH2. If the value of this property is *`NONE`*, the switch device takes no action on existing paths. If the value is *`BREAK_BEFORE_MAKE`*, the switch device breaks conflicting paths before making new ones. If the value is *`BREAK_AFTER_MAKE`*, the switch device breaks conflicting paths after making new ones. Most switch devices support only one of the possible values. In such cases, this property serves as an indicator of the device's behavior.

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	enums.ScanMode
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Scanning Configuration:Scan Mode**
- C Attribute: **NISWITCH_ATTR_SCAN_MODE**

serial_number

`niswitch.Session.`**`serial_number`**

This read-only property returns the serial number for the switch device controlled by this instrument driver. If the device does not return a serial number, the driver returns the IVI_ERROR_ATTRIBUTE_NOT_SUPPORTED error.

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: **Module Characteristics:Serial Number**
- C Attribute: **NISWITCH_ATTR_SERIAL_NUMBER**

settling_time

`niswitch.Session.`**`settling_time`**

This channel-based property returns the maximum length of time from after you make a connection until the signal flowing through the channel settles. The units are seconds. the greater value of the settling time and the value you specify as the scan delay.

Note: NI PXI-2501/2503/2565/2590/2591 Users–the actual delay will always be

Tip: This property can be set/get on specific channels within your *niswitch.Session* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].settling_time`

To set/get on all channels, you can call the property directly on the *niswitch.Session*.

Example: `my_session.settling_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: **Module Characteristics:Settling Time**
- C Attribute: **NISWITCH_ATTR_SETTLING_TIME**

simulate

`niswitch.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 `niswitch.Session.InitWithOptions()` method to override this value.

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	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: **NISWITCH_ATTR_SIMULATE**

specific_driver_description

`niswitch.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: **NISWITCH_ATTR_SPECIFIC_DRIVER_DESCRIPTION**

specific_driver_revision

`niswitch.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: **NISWITCH_ATTR_SPECIFIC_DRIVER_REVISION**

specific_driver_vendor

`niswitch.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: **NISWITCH_ATTR_SPECIFIC_DRIVER_VENDOR**

supported_instrument_models

`niswitch.Session.`**`supported_instrument_models`**

Contains a comma-separated list of supported instrument models.

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: **NISWITCH_ATTR_SUPPORTED_INSTRUMENT_MODELS**

temperature

`niswitch.Session.`**`temperature`**

This property returns the temperature as read by the Switch module. The units are degrees Celsius.

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: **Module Characteristics:Temperature**
- C Attribute: **NISWITCH_ATTR_TEMPERATURE**

trigger_input

`niswitch.Session.`**`trigger_input`**

This property specifies the source of the trigger for which the switch device can wait when processing a scan list. The switch device waits for a trigger when it encounters a semi-colon in a scan list. When the trigger occurs, the switch device advances to the next entry in the scan list.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TriggerInput
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Scanning Configuration:Trigger Input**
- C Attribute: **NISWITCH_ATTR_TRIGGER_INPUT**

trigger_input_polarity

`niswitch.Session.`**`trigger_input_polarity`**

Determines the behavior of the trigger Input.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TriggerInputPolarity
Permissions	read-write
Repeated Capabilities	None

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Scanning Configuration:Trigger Input Polarity**
- C Attribute: **NISWITCH_ATTR_TRIGGER_INPUT_POLARITY**

wire_mode

`niswitch.Session.`**`wire_mode`**

This property returns the wire mode of the switch device. This property affects the values of the *`niswitch.Session.num_of_rows`* and *`niswitch.Session.num_of_columns`* properties. The actual number of input and output lines on the switch device is fixed, but the number of channels depends on how many lines constitute each channel.

Tip: This property can be set/get on specific channels within your *`niswitch.Session`* instance. Use Python index notation on the repeated capabilities container channels to specify a subset.

Example: `my_session.channels[ ... ].wire_mode`

To set/get on all channels, you can call the property directly on the *`niswitch.Session`*.

Example: `my_session.wire_mode`

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: **Module Characteristics:Wire mode**
- C Attribute: **NISWITCH_ATTR_WIRE_MODE**

Session

- *num_of_columns*
- *num_of_rows*
- *power_down_latching_relays_after_debounce*
- *scan_advanced_output*
- *scan_advanced_polarity*
- *scan_delay*
- *scan_list*
- *scan_mode*
- *serial_number*
- *settling_time*
- *simulate*
- *specific_driver_description*
- *specific_driver_revision*
- *specific_driver_vendor*
- *supported_instrument_models*
- *temperature*
- *trigger_input*
- *trigger_input_polarity*
- *wire_mode*

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 `niSwitch_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

`niswitch.Session.channels[]`

```
session.channels['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

Enums

Enums used in NI-SWITCH

HandshakingInitiation

class niswitch.**HandshakingInitiation**

MEASUREMENT_DEVICE
The *niSwitch Initiate Scan <switchviref.chm::/:py:meth:'niswitch.Session.Initiate_Scan*.html>'__ VI does not return until the switch hardware is waiting for a trigger input. This ensures that if you initiate the measurement device after calling the *niSwitch Initiate Scan <switchviref.chm::/:py:meth:'niswitch.Session.Initiate_Scan*.html>'__ VI , the switch is sure to receive the first measurement complete (MC) signal sent by the measurement device. The measurement device should be configured to first take a measurement, send MC, then wait for scanner advanced output signal. Thus, the first MC of the measurement device initiates handshaking.

SWITCH
The *niSwitch Initiate Scan <switchviref.chm::/:py:meth:'niswitch.Session.Initiate_Scan*.html>'__ VI returns immediately after beginning scan list execution. It is assumed that the measurement device has already been configured and is waiting for the scanner advanced signal. The measurement should be configured to first wait for a trigger, then take a measurement. Thus, the first scanner advanced output signal of the switch module initiates handshaking.

PathCapability

class niswitch.**PathCapability**

PATH_AVAILABLE
Path Available

PATH_EXISTS
Path Exists

PATH_UNSUPPORTED
Path Unsupported

RESOURCE_IN_USE
Resource in use

SOURCE_CONFLICT
Source conflict

CHANNEL_NOT_AVAILABLE
Channel not available

RelayAction

class niswitch.**RelayAction**

OPEN
Open Relay

CLOSE
Close Relay

RelayPosition

class niswitch.**RelayPosition**

OPEN
Open

CLOSED
Closed

ScanAdvancedOutput

class niswitch.**ScanAdvancedOutput**

NONE
The switch device does not produce a Scan Advanced Output trigger.

EXTERNAL
External Trigger. The switch device produces the Scan Advanced Output trigger on the external trigger output.

TTL0
The switch device produces the Scan Advanced Output on the PXI TRIG0 line.

TTL1
The switch device produces the Scan Advanced Output on the PXI TRIG1 line.

TTL2
The switch device produces the Scan Advanced Output on the PXI TRIG2 line.

TTL3
The switch device produces the Scan Advanced Output on the PXI TRIG3 line.

TTL4
The switch device produces the Scan Advanced Output on the PXI TRIG4 line.

TTL5
The switch device produces the Scan Advanced Output on the PXI TRIG5 line.

TTL6
The switch device produces the Scan Advanced Output on the PXI TRIG6 line.

TTL7
The switch device produces the Scan Advanced Output on the PXI TRIG7 line.

PXI_STAR
The switch module produces the Scan Advanced Output Trigger on the PXI Star trigger bus before processing the next entry in the scan list.

REARCONNECTOR
The switch device produces the Scan Advanced Output trigger on the rear connector.

FRONTCONNECTOR
The switch device produces the Scan Advanced Output trigger on the front connector.

`REARCONNECTOR_MODULE1`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 1.

`REARCONNECTOR_MODULE2`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 2.

`REARCONNECTOR_MODULE3`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 3.

`REARCONNECTOR_MODULE4`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 4.

`REARCONNECTOR_MODULE5`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 5.

`REARCONNECTOR_MODULE6`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 6.

`REARCONNECTOR_MODULE7`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 7.

`REARCONNECTOR_MODULE8`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 8.

`REARCONNECTOR_MODULE9`
The switch module produces the Scan Advanced Ouptut Trigger on the rear connector module 9.

`REARCONNECTOR_MODULE10`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 10.

`REARCONNECTOR_MODULE11`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 11.

`REARCONNECTOR_MODULE12`
The switch module produces the Scan Advanced Output Trigger on the rear connector module 12.

`FRONTCONNECTOR_MODULE1`
The switch module produces the Scan Advanced Output Trigger on the front connector module 1.

`FRONTCONNECTOR_MODULE2`
The switch module produces the Scan Advanced Output Trigger on the front connector module 2.

`FRONTCONNECTOR_MODULE3`
The switch module produces the Scan Advanced Output Trigger on the front connector module 3.

`FRONTCONNECTOR_MODULE4`
The switch module produces the Scan Advanced Output Trigger on the front connector module 4.

`FRONTCONNECTOR_MODULE5`
The switch module produces the Scan Advanced Output Trigger on the front connector module 5.

`FRONTCONNECTOR_MODULE6`
The switch module produces the Scan Advanced Output Trigger on the front connector module 6.

`FRONTCONNECTOR_MODULE7`
The switch module produces the Scan Advanced Output Trigger on the front connector module 7.

`FRONTCONNECTOR_MODULE8`
The switch module produces the Scan Advanced Output Trigger on the front connector module 8.

`FRONTCONNECTOR_MODULE9`
The switch module produces the Scan Advanced Output Trigger on the front connector module 9.

FRONTCONNECTOR_MODULE10
The switch module produces the Scan Advanced Output Trigger on the front connector module 10.

FRONTCONNECTOR_MODULE11
The switch module produces the Scan Advanced Output Trigger on the front connector module 11.

FRONTCONNECTOR_MODULE12
The switch module produces the Scan Advanced Output Trigger on the front connector module 12.

ScanAdvancedPolarity

class `niswitch.`**`ScanAdvancedPolarity`**

RISING
The trigger occurs on the rising edge of the signal.

FALLING
The trigger occurs on the falling edge of the signal.

ScanMode

class `niswitch.`**`ScanMode`**

NONE
No implicit action on connections when scanning.

BREAK_BEFORE_MAKE
When scanning, the switch device breaks existing connections before making new connections.

BREAK_AFTER_MAKE
When scanning, the switch device breaks existing connections after making new connections.

TriggerInput

class `niswitch.`**`TriggerInput`**

IMMEDIATE
Immediate Trigger. The switch device does not wait for a trigger before processing the next entry in the scan list.

EXTERNAL
External Trigger. The switch device waits until it receives a trigger from an external source through the external trigger input before processing the next entry in the scan list.

SOFTWARE_TRIG
The switch device waits until you call the *`niswitch.Session.send_software_trigger()`* method before processing the next entry in the scan list.

TTL0
The switch device waits until it receives a trigger on the PXI TRIG0 line before processing the next entry in the scan list.

TTL1
The switch device waits until it receives a trigger on the PXI TRIG1 line before processing the next entry in the scan list.

TTL2
The switch device waits until it receives a trigger on the PXI TRIG2 line before processing the next entry in the scan list.

TTL3
The switch device waits until it receives a trigger on the PXI TRIG3 line before processing the next entry in the scan list.

TTL4
The switch device waits until it receives a trigger on the PXI TRIG4 line before processing the next entry in the scan list.

TTL5
The switch device waits until it receives a trigger on the PXI TRIG5 line before processing the next entry in the scan list.

TTL6
The switch device waits until it receives a trigger on the PXI TRIG6 line before processing the next entry in the scan list.

TTL7
The switch device waits until it receives a trigger on the PXI TRIG7 line before processing the next entry in the scan list.

PXI_STAR
The switch device waits until it receives a trigger on the PXI STAR trigger bus before processing the next entry in the scan list.

REARCONNECTOR
The switch device waits until it receives a trigger on the rear connector.

FRONTCONNECTOR
The switch device waits until it receives a trigger on the front connector.

REARCONNECTOR_MODULE1
The switch module waits until it receives a trigger on the rear connector module 1.

REARCONNECTOR_MODULE2
The switch module waits until it receives a trigger on the rear connector module 2.

REARCONNECTOR_MODULE3
The switch module waits until it receives a trigger on the rear connector module 3.

REARCONNECTOR_MODULE4
The switch module waits until it receives a trigger on the rear connector module 4.

REARCONNECTOR_MODULE5
The switch module waits until it receives a trigger on the rear connector module 5.

REARCONNECTOR_MODULE6
The switch module waits until it receives a trigger on the rear connector module 6.

REARCONNECTOR_MODULE7
The switch module waits until it receives a trigger on the rear connector module 7.

REARCONNECTOR_MODULE8
The switch module waits until it receives a trigger on the rear connector module 8.

REARCONNECTOR_MODULE9
The switch module waits until it receives a trigger on the rear connector module 9.

REARCONNECTOR_MODULE10
The switch module waits until it receives a trigger on the rear connector module 10.

REARCONNECTOR_MODULE11
The switch module waits until it receives a trigger on the rear connector module 11.

REARCONNECTOR_MODULE12
The switch module waits until it receives a trigger on the rear connector module 12.

FRONTCONNECTOR_MODULE1
The switch module waits until it receives a trigger on the front connector module 1.

FRONTCONNECTOR_MODULE2
The switch module waits until it receives a trigger on the front connector module 2.

FRONTCONNECTOR_MODULE3
The switch module waits until it receives a trigger on the front connector module 3.

FRONTCONNECTOR_MODULE4
The switch module waits until it receives a trigger on the front connector module 4.

FRONTCONNECTOR_MODULE5
The switch module waits until it receives a trigger on the front connector module 5.

FRONTCONNECTOR_MODULE6
The switch module waits until it receives a trigger on the front connector module 6.

FRONTCONNECTOR_MODULE7
The switch module waits until it receives a trigger on the front connector module 7.

FRONTCONNECTOR_MODULE8
The switch module waits until it receives a trigger on the front connector module 8.

FRONTCONNECTOR_MODULE9
The switch module waits until it receives a trigger on the front connector module 9.

FRONTCONNECTOR_MODULE10
The switch module waits until it receives a trigger on the front connector module 10.

FRONTCONNECTOR_MODULE11
The switch module waits until it receives a trigger on the front connector module 11.

FRONTCONNECTOR_MODULE12
The switch module waits until it receives a trigger on the front connector module 12.

TriggerInputPolarity

class niswitch.**TriggerInputPolarity**

RISING
The trigger occurs on the rising edge of the signal.

FALLING
The trigger occurs on the falling edge of the signal.

Exceptions and Warnings

Error

exception `niswitch.errors.`**`Error`**

Base exception type that all NI-SWITCH exceptions derive from

DriverError

exception `niswitch.errors.`**`DriverError`**

An error originating from the NI-SWITCH driver

UnsupportedConfigurationError

exception `niswitch.errors.`**`UnsupportedConfigurationError`**

An error due to using this module in an usupported platform.

DriverNotInstalledError

exception `niswitch.errors.`**`DriverNotInstalledError`**

An error due to using this module without the driver runtime installed.

InvalidRepeatedCapabilityError

exception `niswitch.errors.`**`InvalidRepeatedCapabilityError`**

An error due to an invalid character in a repeated capability

SelfTestError

exception `niswitch.errors.`**`SelfTestError`**

An error due to a failed self-test

DriverWarning

exception `niswitch.errors.`**`DriverWarning`**

A warning originating from the NI-SWITCH driver

Examples

You can download all niswitch examples here

niswitch_connect_channels.py

Listing 17: (niswitch_connect_channels.py)

```python
#!/usr/bin/python

import argparse
import niswitch
import sys


def example(resource_name, channel1, channel2, topology, simulate):
    # if we are simulating resource name must be blank
    resource_name = '' if simulate else resource_name

    with niswitch.Session(resource_name=resource_name, topology=topology,␣
↪simulate=simulate) as session:
        session.connect(channel1=channel1, channel2=channel2)
        print('Channel ', channel1, ' and ', channel2, ' are now connected.')
        session.disconnect(channel1=channel1, channel2=channel2)
        print('Channel ', channel1, ' and ', channel2, ' are now disconnected.')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Performs a connection with NI-
↪SWITCH Channels.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of a National Instruments Switch.')
    parser.add_argument('-ch1', '--channel1', default='c0', help='Channel One.')
    parser.add_argument('-ch2', '--channel2', default='r0', help='Channel Two.')
    parser.add_argument('-t', '--topology', default='Configured Topology', help=
↪'Topology.')
    parser.add_argument('-s', '--simulate', default=False, action='store_true', help=
↪'Simulate device.')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.channel1, args.channel2, args.topology, args.
↪simulate)


def test_example():
    example('', 'c0', 'r0', '2737/2-Wire 4x64 Matrix', True)


def test_main():
    cmd_line = ['--topology', '2737/2-Wire 4x64 Matrix', '--simulate']
    _main(cmd_line)


def main():
    _main(sys.argv[1:])


if __name__ == '__main__':
    main()


```

niswitch_get_device_info.py

Listing 18: (niswitch_get_device_info.py)

```
#!/usr/bin/python

import argparse
import niswitch
import sys


def example(resource_name, topology, simulate, device, channel, relay):
    # if we are simulating resource name must be blank
    resource_name = '' if simulate else resource_name

    with niswitch.Session(resource_name=resource_name, topology=topology,␣
↪simulate=simulate) as session:
        if device:
            print('Device Info:')
            row_format = '{:<18}' * (2)
            print(row_format.format('Device Name: ', session.io_resource_descriptor))
            print(row_format.format('Device Model: ', session.instrument_model))
            print(row_format.format('Driver Revision: ', session.specific_driver_
↪revision))
            print(row_format.format('Channel count: ', session.channel_count))
            print(row_format.format('Relay count: ', session.number_of_relays))
        if channel:
            print('Channel Info:')
            row_format = '{:6}' + ' ' * 12 + '{:<15}{:<22}{:6}'
            print(row_format.format('Number', 'Name', 'Is Configuration', 'Is Source
↪'))
            for i in range(1, session.channel_count + 1):
                channel_name = session.get_channel_name(index=i)
                channel = session.channels[channel_name]
                print(row_format.format(i, channel_name, str(channel.is_configuration_
↪channel), str(channel.is_source_channel)))
        if relay:
            print('Relay Info:')
            row_format = '{:6}' + ' ' * 12 + '{:<15}{:<22}{:6}'
            print(row_format.format('Number', 'Name', 'Position', 'Count'))
            for i in range(1, session.number_of_relays + 1):
                relay_name = session.get_relay_name(index=i)
                print(row_format.format(i, relay_name, session.get_relay_
↪position(relay_name=relay_name), session.get_relay_count(relay_name=relay_name)))


def _main(argsv):
    parser = argparse.ArgumentParser(description='Prints information for the␣
↪specified National Instruments Switch module.', formatter_class=argparse.
↪ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of a National Instruments Switch.')
    parser.add_argument('-d', '--device', default=False, action='store_true', help=
↪'Prints information for the device')
    parser.add_argument('-c', '--channel', default=False, action='store_true', help=
↪'Prints information for all channels on the device')
    parser.add_argument('-r', '--relay', default=False, action='store_true', help=
↪'Prints information for all relays on the device')
    parser.add_argument('-t', '--topology', default='Configured Topology', help=
↪'Topology.')
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('-s', '--simulate', default=False, action='store_true', help=
↪'Simulate device.')
    args = parser.parse_args(argsv)

    if not (args.device or args.channel or args.relay):
        print('You must specify at least one of -d, -c, or -r!')
        parser.print_help()
        sys.exit(1)

    example(args.resource_name, args.topology, args.simulate, args.device, args.
↪channel, args.relay)


def test_example():
    example('', '2737/2-Wire 4x64 Matrix', True, True, True, True)


def test_main():
    cmd_line = ['--topology', '2737/2-Wire 4x64 Matrix', '--simulate', '--device', '--
↪channel', '--relay', ]
    _main(cmd_line)


def main():
    _main(sys.argv[1:])


if __name__ == '__main__':
    main()


```

niswitch_relay_control.py

Listing 19: (niswitch_relay_control.py)

```
#!/usr/bin/python

import argparse
import niswitch
import sys


def example(resource_name, topology, simulate, relay, action):
    # if we are simulating resource name must be blank
    resource_name = '' if simulate else resource_name

    with niswitch.Session(resource_name=resource_name, topology=topology,␣
↪simulate=simulate) as session:
        session.relay_control(relay_name=relay, relay_action=niswitch.
↪RelayAction[action])
        print('Relay ', relay, ' has had the action ', action, ' performed.')


```

(continues on next page)

(continued from previous page)

```
def _main(argsv):
    parser = argparse.ArgumentParser(description='Performs relay control with NI-
↪SWITCH relays.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
↪name of a National Instruments Switch.')
    parser.add_argument('-r', '--relay', default='k0', help='Relay Name.')
    parser.add_argument('-a', '--action', default='OPEN', choices=niswitch.
↪RelayAction.__members__.keys(), type=str.upper, help='Relay Action.')
    parser.add_argument('-t', '--topology', default='Configured Topology', help=
↪'Topology.')
    parser.add_argument('-s', '--simulate', default=False, action='store_true', help=
↪'Simulate device.')
    args = parser.parse_args(argsv)
    example(args.resource_name, args.topology, args.simulate, args.relay, args.action)


def test_example():
    example('', '2737/2-Wire 4x64 Matrix', True, 'kr0c0', 'OPEN')


def test_main():
    cmd_line = ['--topology', '2737/2-Wire 4x64 Matrix', '--simulate', '--relay',
↪'kr0c0']
    _main(cmd_line)


def main():
    _main(sys.argv[1:])


if __name__ == '__main__':
    main()


```

7.7 nise module

7.7.1 Installation

As a prerequisite to using the nise module, you must install the NI Switch Executive runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

The nimi-python modules (i.e. for **NI Switch Executive**) can be installed with pip:

```
$ python -m pip install nise~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install nise
```

7.7.2 Usage

The following is a basic example of using the **nise** module to open a session to a Switch Executive Virtual Device and connect a routegroup.

```
import nise
with nise.Session('SwitchExecutiveExample') as session:
    session.connect('DIOToUUT')
```

Additional examples for NI Switch Executive are located in src/nise/examples/ directory.

7.7.3 API Reference

Session

class nise.**Session**(*self*, *virtual_device_name*, *options={}*)

Opens a session to a specified NI Switch Executive virtual device. Opens communications with all of the IVI switches associated with the specified NI Switch Executive virtual device. Returns a session handle that you use to identify the virtual device in all subsequent NI Switch Executive method calls. NI Switch Executive uses a reference counting scheme to manage open session handles to an NI Switch Executive virtual device. Each call to `nise.Session.__init__()` must be matched with a subsequent call to *`nise.Session.close()`*. Successive calls to `nise.Session.__init__()` with the same virtual device name always returns the same session handle. NI Switch Executive disconnects its communication with the IVI switches after all session handles are closed to a given virtual device. The session handles may be used safely in multiple threads of an application. Sessions may only be opened to a given NI Switch Executive virtual device from a single process at a time.

Parameters

- **virtual_device_name** (*str*) – The name of the NI Switch Executive virtual device.
- **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	{}

Methods

close

`nise.Session.close()`

Reduces the reference count of open sessions by one. If the reference count goes to 0, the method deallocates any memory resources the driver uses and closes any open IVI switch sessions. After calling the `nise.Session.close()` method, you should not use the NI Switch Executive virtual device again until you call `nise.Session.__init__()`.

Note: This method is not needed when using the session context manager

connect

`nise.Session.connect(`*connect_spec*, *multiconnect_mode=nise.MulticonnectMode.DEFAULT*, *wait_for_debounce=True*`)`

Connects the routes specified by the connection specification. When connecting, it may allow for multiconnection based on the multiconnection mode. In the event of an error, the call to `nise.Session.connect()` will attempt to undo any connections made so that the system will be left in the same state that it was in before the call was made. Some errors can be caught before manipulating hardware, although it is feasible that a hardware call could fail causing some connections to be momentarily closed and then reopened. If the wait for debounce parameter is set, the method will not return until the switch system has debounced.

Parameters

- **connect_spec** (*str*) – String describing the connections to be made. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.
- **multiconnect_mode** (*nise.MulticonnectMode*) – This value sets the connection mode for the method. The mode might be one of the following: `NISE_VAL_USE_DEFAULT_MODE` (-1) - uses the mode selected as the default for the route in the NI Switch Executive virtual device configuration. If a mode has not been selected for the route in the NI Switch Executive virtual device, this parameter defaults to `NISE_VAL_MULTICONNECT_ROUTES`. *NO_MULTICONNECT* (0) - routes specified in the connection specification must be disconnected before they can be reconnected. Calling Connect on a route that was connected using No Multiconnect mode results in an error condition. `NISE_VAL_MULTICONNECT_ROUTES` (1)- routes specified in the connection specification can be connected multiple times. The first call to Connect performs the physical hardware connection. Successive calls to Connect increase a connection reference count. Similarly, calls to Disconnect decrease the reference count. Once it reaches 0, the hardware is physically disconnected. Multiconnecting routes applies to entire routes and not to route segments.

  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.

- **wait_for_debounce** (*bool*) – Waits (if true) for switches to debounce between its connect and disconnect operations. If false, it immediately begins the second operation after completing the first. The order of connect and disconnect operation is set by the Operation Order input.

connect_and_disconnect

nise.Session.**connect_and_disconnect**(*connect_spec*, *disconnect_spec*, *multiconnect_mode=nise.MulticonnectMode.DEFAULT*, *operation_order=nise.OperationOrder.AFTER*, *wait_for_debounce=True*)

Connects routes and disconnects routes in a similar fashion to `nise.Session.connect()` and `nise.Session.disconnect()` except that the operations happen in the context of a single method call. This method is useful for switching from one state to another state. `nise.Session.connect_and_disconnect()` manipulates the hardware connections and disconnections only when the routes are different between the connection and disconnection specifications. If any routes are common between the connection and disconnection specifications, NI Switch Executive determines whether or not the relays need to be switched. This functionality has the distinct advantage of increased throughput for shared connections, because hardware does not have to be involved and potentially increases relay lifetime by decreasing the number of times that the relay has to be switched. In the event of an error, the call to `nise.Session.connect_and_disconnect()` attempts to undo any connections made, but does not attempt to reconnect disconnections. Some errors can be caught before manipulating hardware, although it is feasible that a hardware call could fail causing some connections to be momentarily closed and then reopened.

Parameters

- **connect_spec** (*str*) – String describing the connections to be made. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.
- **disconnect_spec** (*str*) – String describing the disconnections to be made. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.
- **multiconnect_mode** (*nise.MulticonnectMode*) – This value sets the connection mode for the method. The mode might be one of the following: `NISE_VAL_USE_DEFAULT_MODE` (-1) - uses the mode selected as the default for the route in the NI Switch Executive virtual device configuration. If a mode has not been selected for the route in the NI Switch Executive virtual device, this parameter defaults to `NISE_VAL_MULTICONNECT_ROUTES`. `NO_MULTICONNECT` (0) - routes specified in the connection specification must be disconnected before they can be reconnected. Calling Connect on a route that was connected using No Multiconnect mode results in an error condition. `NISE_VAL_MULTICONNECT_ROUTES` (1) - routes specified in the connection specification can be connected multiple times. The first call to Connect performs the physical hardware connection. Successive calls to Connect increase a connection reference count. Similarly, calls to

Disconnect decrease the reference count. Once it reaches 0, the hardware is physically disconnected. This behavior is slightly different with SPDT relays. For more information, refer to the Exclusions and SPDT Relays topic in the NI Switch Executive Help. Multiconnecting routes applies to entire routes and not to route segments.

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.

- **`operation_order`** (*`nise.OperationOrder`*) – Sets the order of the operation for the method. Defined values are Break Before Make and Break After Make. *`BEFORE`* (1) - The method disconnects the routes specified in the disconnect specification before connecting the routes specified in the connect specification. This is the typical mode of operation. *`AFTER`* (2) - The method connects the routes specified in the connection specification before connecting the routes specified in the disconnection specification. This mode of operation is normally used when you are switching current and want to ensure that a load is always connected to your source. The order of operation is to connect first or disconnect first.
- **`wait_for_debounce`** (*`bool`*) – Waits (if true) for switches to debounce between its connect and disconnect operations. If false, it immediately begins the second operation after completing the first. The order of connect and disconnect operation is set by the Operation Order input.

disconnect

`nise.Session.`**`disconnect`**`(`*disconnect_spec*`)`

Disconnects the routes specified in the Disconnection Specification. If any of the specified routes were originally connected in a multiconnected mode, the call to *`nise.Session.disconnect()`* reduces the reference count on the route by 1. If the reference count reaches 0, it is disconnected. If a specified route does not exist, it is an error condition. In the event of an error, the call to *`nise.Session.disconnect()`* continues to try to disconnect everything specified by the route specification string but reports the error on completion.

Parameters **`disconnect_spec`** (*`str`*) – String describing the disconnections to be made. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.

disconnect_all

`nise.Session.`**`disconnect_all`**`()`

Disconnects all connections on every IVI switch device managed by the NISE session reference passed to this method. *`nise.Session.disconnect_all()`* ignores all multiconnect modes. Calling *`nise.Session.disconnect_all()`* resets all of the switch states for the system.

expand_route_spec

nise.Session.**expand_route_spec**(*route_spec, expand_action=nise.ExpandAction.ROUTES, expanded_route_spec_size=[1024]*)

Expands a route spec string to yield more information about the routes and route groups within the spec. The route specification string returned from *nise.Session.expand_route_spec()* can be passed to other Switch Executive API methods (such as *nise.Session.connect()*, *nise.Session.disconnect()*, and *nise.Session.connect_and_disconnect()*) that use route specification strings.

Parameters

- **route_spec** (*str*) – String describing the routes and route groups to expand. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.
- **expand_action** (*nise.ExpandAction*) – This value sets the expand action for the method. The action might be one of the following: *ROUTES* (0) - expands the route spec to routes. Converts route groups to their constituent routes. *PATHS* (1) - expands the route spec to paths. Converts routes and route groups to their constituent square bracket route spec strings. Example: [Dev1/c0->Dev1/r0->Dev1/c1]
- **expanded_route_spec_size** (*list of int*) – The routeSpecSize is an ViInt32 that is passed by reference into the method. As an input, it is the size of the route spec string buffer being passed. If the route spec string is larger than the string buffer being passed, only the portion of the route spec string that can fit in the string buffer is copied into it. On return from the method, routeSpecSize holds the size required to hold the entire route spec string. Note that this size may be larger than the buffer size as the method always returns the size needed to hold the entire buffer. You may pass NULL for this parameter if you are not interested in the return value for routeSpecSize and routeSpec.

Return type str

Returns The expanded route spec. Route specification strings can be directly passed to *nise.Session.connect()*, *nise.Session.disconnect()*, or *nise.Session.connect_and_disconnect()* Refer to Route Specification Strings in the NI Switch Executive Help for more information. You may pass NULL for this parameter if you are not interested in the return value. To obtain the route specification string, you should pass a buffer to this parameter. The size of the buffer required may be obtained by calling the method with NULL for this parameter and a valid ViInt32 to routeSpecSize. The routeSpecSize will contain the size needed to hold the entire route specification (including the NULL termination character). Common operation is to call the method twice. The first time you call the method you can determine the size needed to hold the route specification string. Allocate a buffer of the appropriate size and then re-call the method to obtain the entire buffer.

find_route

nise.Session.**find_route**(*channel1, channel2, route_spec_size=[1024]*)

Finds an existing or potential route between channel 1 and channel 2. The returned route specification contains the route specification and the route capability determines whether or not the route existed, is possible, or is not possible for various reasons. The route specification string returned from `nise.Session.find_route()` can be passed to other Switch Executive API methods (such as `nise.Session.connect()`, `nise.Session.disconnect()`, and `nise.Session.connect_and_disconnect()`) that use route specification strings.

Parameters

- **channel1** (*str*) – Channel name of one of the endpoints of the route to find. The channel name must either be a channel alias name or a name in the device/ivichannel syntax. Examples: MyChannel Switch1/R0
- **channel2** (*str*) – Channel name of one of the endpoints of the route to find. The channel name must either be a channel alias name or a name in the device/ivichannel syntax. Examples: MyChannel Switch1/R0
- **route_spec_size** (*list of int*) – The routeSpecSize is an ViInt32 that is passed by reference into the method. As an input, it is the size of the route string buffer being passed. If the route string is larger than the string buffer being passed, only the portion of the route string that can fit in the string buffer is copied into it. On return from the method, routeSpecSize holds the size required to hold the entire route string. Note that this size may be larger than the buffer size as the method always returns the size needed to hold the entire buffer. You may pass NULL for this parameter if you are not interested in the return value for routeSpecSize and routeSpec.

Return type

tuple (route_spec, path_capability)

WHERE

route_spec (str):

The fully specified route path complete with delimiting square brackets if the route exists or is possible. An example of a fully specified route string is: [A->Switch1/r0->B] Route specification strings can be directly passed to `nise.Session.connect()`, `nise.Session.disconnect()`, or `nise.Session.connect_and_disconnect()` Refer to Route Specification Strings in the NI Switch Executive Help for more information. You may pass NULL for this parameter if you are not interested in the return value. To obtain the route specification string, you should pass a buffer to this parameter. The size of the buffer required may be obtained by calling the method with NULL for this parameter and a valid ViInt32 to routeSpecSize. The routeSpecSize will contain the size needed to hold the entire route specification (including the NULL termination character). Common operation is to call the method twice. The first time you call the method you can determine the size needed to hold the route specification string. Allocate a buffer of the appropriate size and then re-call the method to obtain the entire buffer.

path_capability (`nise.PathCapability`):

The return value which expresses the capability of finding a valid route between Channel 1 and Channel 2. Refer to the table below for value descriptions. You may pass NULL for this parameter if you are not interested in the return value. Route capability might be one of the following: Path Available (1) A path between channel 1 and channel 2 is available. The route specification parameter returns a string describing the available path. Path Exists (2) A path between channel 1 and channel 2 already exists. The route specification parameter returns a string describing the existing path. Path Unsupported (3) There is no potential path between channel 1 and channel 2 given the current configuration. Resource In Use (4) There is a potential path between channel 1 and channel 2, although a resource needed to complete the path is already in use. Source Conflict (5) Channel 1 and channel 2 cannot be connected because their connection would result in an exclusion violation. Channel Not Available (6) One of the channels is not useable as an endpoint channel. Make sure that it is not marked as a reserved for routing. Channels Hardwired (7) The two channels reside on the same hardwire. An implicit path already exists.

get_all_connections

`nise.Session.`**`get_all_connections`**`(`*route_spec_size=[1024]*`)`

Returns the top-level connected routes and route groups. The route specification string returned from *`nise.Session.get_all_connections()`* can be passed to other Switch Executive API methods (such as *`nise.Session.connect()`*, *`nise.Session.disconnect()`*, *`nise.Session.connect_and_disconnect()`*, and *`nise.Session.expand_route_spec()`*) that use route specification strings.

Parameters **`route_spec_size`** (*`list of int`*) – The routeSpecSize is an ViInt32 that is passed by reference into the method. As an input, it is the size of the route spec string buffer being passed. If the route spec string is larger than the string buffer being passed, only the portion of the route spec string that can fit in the string buffer is copied into it. On return from the method, routeSpecSize holds the size required to hold the entire route spec string. Note that this size may be larger than the buffer size as the method always returns the size needed to hold the entire buffer. You may pass NULL for this parameter if you are not interested in the return value for routeSpecSize and routeSpec.

Return type str

Returns The route spec of all currently connected routes and route groups. Route specification strings can be directly passed to *`nise.Session.connect()`*, *`nise.Session.disconnect()`*, *`nise.Session.connect_and_disconnect()`*, or *`nise.Session.expand_route_spec()`* Refer to Route Specification Strings in the NI Switch Executive Help for more information. You may pass NULL for this parameter if you are not interested in the return value. To obtain the route specification string, you should pass a buffer to this parameter. The size of the buffer required may be obtained by calling the method with NULL for this parameter and a valid ViInt32 to routeSpecSize. The routeSpecSize will contain the size needed to hold the entire route specification (including the NULL termination character). Common operation is to call the method twice. The first time you call the method you can determine the size needed to hold the route specification string. Allocate a buffer of the appropriate size and then re-call the method to obtain the entire buffer.

is_connected

`nise.Session.`**`is_connected`**`(`*route_spec*`)`

Checks whether the specified routes and routes groups are connected. It returns true if connected.

Parameters **`route_spec`** (*`str`*) – String describing the connections to check. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are:

MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.

Return type bool

Returns Returns TRUE if the routes and routes groups are connected or FALSE if they are not.

is_debounced

`nise.Session.is_debounced()`

Checks to see if the switching system is debounced or not. This method does not wait for debouncing to occur. It returns true if the system is fully debounced. This method is similar to the IviSwtch specific method.

Return type bool

Returns Returns TRUE if the system is fully debounced or FALSE if it is still settling.

wait_for_debounce

`nise.Session.wait_for_debounce(`*maximum_time_ms=hightime.timedelta(milliseconds=-1)*`)`

Waits for all of the switches in the NI Switch Executive virtual device to debounce. This method does not return until either the switching system is completely debounced and settled or the maximum time has elapsed and the system is not yet debounced. In the event that the maximum time elapses, the method returns an error indicating that a timeout has occurred. To ensure that all of the switches have settled, NI recommends calling `nise.Session.wait_for_debounce()` after a series of connection or disconnection operations and before taking any measurements of the signals connected to the switching system.

Parameters **`maximum_time_ms`** (*`hightime.timedelta`, `datetime.timedelta`, `or int in milliseconds`*) – The amount of time to wait (in milliseconds) for the debounce to complete. A value of 0 checks for debouncing once and returns an error if the system is not debounced at that time. A value of -1 means to block for an infinite period of time until the system is debounced.

Session

Enums

Enums used in NI Switch Executive

ExpandAction

class nise.**ExpandAction**

ROUTES
Expand to routes

PATHS
Expand to paths

MulticonnectMode

class nise.**MulticonnectMode**

DEFAULT
Default

NO_MULTICONNECT
No multiconnect

MULTICONNECT
Multiconnect

OperationOrder

class nise.**OperationOrder**

BEFORE
Break before make

AFTER
Break after make

PathCapability

class nise.**PathCapability**

PATH_NEEDS_HARDWIRE
Path needs hardwire

PATH_NEEDS_CONFIG_CHANNEL
Path needs config channel

PATH_AVAILABLE
Path available

PATH_EXISTS
Path exists

PATH_UNSUPPORTED
Path Unsupported

RESOURCE_IN_USE
Resource in use

EXCLUSION_CONFLICT
Exclusion conflict

CHANNEL_NOT_AVAILABLE
Channel not available

CHANNELS_HARDWIRED
Channels hardwired

Exceptions and Warnings

Error

exception `nise.errors.`**Error**
Base exception type that all NI Switch Executive exceptions derive from

DriverError

exception `nise.errors.`**DriverError**
An error originating from the NI Switch Executive driver

UnsupportedConfigurationError

exception `nise.errors.`**UnsupportedConfigurationError**
An error due to using this module in an usupported platform.

DriverNotInstalledError

exception `nise.errors.`**DriverNotInstalledError**
An error due to using this module without the driver runtime installed.

InvalidRepeatedCapabilityError

exception `nise.errors.`**InvalidRepeatedCapabilityError**
An error due to an invalid character in a repeated capability

DriverWarning

exception nise.errors.**DriverWarning**

A warning originating from the NI Switch Executive driver

Examples

You can download all nise examples here

nise_basic_example.py

Listing 20: (nise_basic_example.py)

```python
#!/usr/bin/python
import argparse
import nise
import sys


def example(virtual_device_name, connection):
    with nise.Session(virtual_device_name=virtual_device_name) as session:
        session.connect(connection)
        print(connection, ' is now connected.')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Connects the specified connection␣
↪specification', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--virtual-device', default='SwitchExecutiveExample',␣
↪help='NI Switch Executive Virtual Device name')
    parser.add_argument('-c', '--connection', default='DIOToUUT', help='Connection␣
↪Specification')
    args = parser.parse_args(argsv)
    example(args.virtual_device, args.connection)


def main():
    _main(sys.argv[1:])


def test_example():
    example('SwitchExecutiveExample', 'DIOToUUT')


def test_main():
    cmd_line = []
    _main(cmd_line)


if __name__ == '__main__':
    main()


```

7.8 nimodinst module

7.8.1 Installation

As a prerequisite to using the nimodinst module, you must install the NI-ModInst runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

The nimi-python modules (i.e. for **NI-ModInst**) can be installed with pip:

```
$ python -m pip install nimodinst~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install nimodinst
```

7.8.2 Usage

The following is a basic example of using the **nimodinst** module to retrieve information on all High Speed Digitizers currently in the system.

```
import nimodinst
with nimodinst.Session("niscope") as session:
    for device in session:
        print("{: >20} {: >15} {: >10}".format(device.device_name, device.device_
↪model, device.serial_number))
```

Additional examples for NI-ModInst are located in src/nimodinst/examples/ directory.

7.8.3 API Reference

Session

class nimodinst.**Session**(*self*, *driver*)

Creates a handle to a list of installed devices supported by the specified driver. Call this method and pass in the name of a National Instruments instrument driver, such as “NI-SCOPE”. This method searches the system and constructs a list of all the installed devices that are supported by that driver, and then returns both a handle to this list and the number of devices found. The handle is used with other methods to query for properties such as device name and model, and to safely discard the list when finished. Note This handle reflects the system state when the handle is created (that is, when you call this method. If you remove devices from the system or rename them in Measurement & Automation Explorer (MAX), this handle may not refer to an accurate list of devices. You should destroy the handle using `nimodinst.Session._close_installed_devices_session()` and create a new handle using this method.

Parameters **driver** (*str*) – A string specifying the driver whose supported devices you want to find. This string is not case-sensitive. Some examples are: NI-SCOPE niScope NI-FGEN niFgen NI-HSDIO niHSDIO NI-DMM niDMM NI-SWITCH niSwitch Note If you use the empty string for this parameter, NI-ModInst creates a list of all Modular Instruments devices installed in the system.

Methods

close

`nimodinst.Session.close()`

Cleans up the NI-ModInst session created by a call to `nimodinst.Session._open_installed_devices_session()`. Call this method when you are finished using the session handle and do not use this handle again.

Note: This method is not needed when using the session context manager

Properties

bus_number

`nimodinst.Session.bus_number`

The bus on which the device has been enumerated.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_BUS_NUMBER**

chassis_number

`nimodinst.Session.chassis_number`

The number of the chassis in which the device is installed. This property can only be queried for PXI devices installed in a chassis that has been properly identified in MAX.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_CHASSIS_NUMBER**

device_model

`nimodinst.Session.device_model`

The model of the device (for example, NI PXI-5122)

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_DEVICE_MODEL**

device_name

`nimodinst.Session.`**`device_name`**

The name of the device, which can be used to open an instrument driver session for that device

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_DEVICE_NAME**

max_pciexpress_link_width

`nimodinst.Session.`**`max_pciexpress_link_width`**

MAX_PCIEXPRESS_LINK_WIDTH

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_MAX_PCIEXPRESS_LINK_WIDTH**

pciexpress_link_width

`nimodinst.Session.`**`pciexpress_link_width`**

PCIEXPRESS_LINK_WIDTH

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_PCIEXPRESS_LINK_WIDTH**

serial_number

`nimodinst.Session.`**`serial_number`**

The serial number of the device

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_SERIAL_NUMBER**

slot_number

`nimodinst.Session.`**`slot_number`**

The slot (for example, in a PXI chassis) in which the device is installed. This property can only be queried for PXI devices installed in a chassis that has been properly identified in MAX.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_SLOT_NUMBER**

socket_number

`nimodinst.Session.`**`socket_number`**

The socket number on which the device has been enumerated

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- C Attribute: **NIMODINST_ATTR_SOCKET_NUMBER**

Session

Exceptions and Warnings

Error

exception `nimodinst.errors.`**`Error`**
Base exception type that all NI-ModInst exceptions derive from

DriverError

exception `nimodinst.errors.`**`DriverError`**
An error originating from the NI-ModInst driver

UnsupportedConfigurationError

exception `nimodinst.errors.`**`UnsupportedConfigurationError`**
An error due to using this module in an usupported platform.

DriverNotInstalledError

exception nimodinst.errors.**DriverNotInstalledError**
An error due to using this module without the driver runtime installed.

DriverWarning

exception nimodinst.errors.**DriverWarning**
A warning originating from the NI-ModInst driver

Examples

You can download all nimodinst examples here

nimodinst_all_devices.py

Listing 21: (nimodinst_all_devices.py)

```python
#!/usr/bin/python

import nimodinst


def example():
    with nimodinst.Session('') as session:
        if len(session) > 0:
            print("%d items" % len(session))
            print("{: >20} {: >15} {: >10}".format('Name', 'Model', 'S/N'))
        for d in session:
            print("{: >20} {: >15} {: >10}".format(d.device_name, d.device_model, d.
↪serial_number))


def _main():
    example()


def test_example():
    example()


if __name__ == '__main__':
    _main()


```

7.9 nitclk module

7.9.1 Installation

As a prerequisite to using the nitclk module, you must install the NI-TClk runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

The nimi-python modules (i.e. for **NI-TClk**) can be installed with pip:

```
$ python -m pip install nitclk~=1.4.0
```

Or **easy_install** from setuptools:

```
$ python -m easy_install nitclk
```

7.9.2 Usage

The following is a basic example of using the **nitclk** module

```
import nitclk
```

Additional examples for NI-TClk are located in src/nitclk/examples/ directory.

7.9.3 API Reference

Public API

The *nitclk* module provides synchronization facilites to allow multiple instruments to simultaneously respond to triggers, to align Sample Clocks on multiple instruments, and/or to simultaneously start multiple instruments.

It consists of a set of functions that act on a list of `SessionReference` objects or instrument *Session* objects for drivers that support NI-TClk. `SessionReference` also has a set of properties for configuration.

```
with niscope.Session('dev1') as scope1, niscope.Session('dev2') as scope2:
    nitclk.configure_for_homogeneous_triggers([scope1, scope2])
    nitclk.initiate([scope1, scope2])
    wfm1 = scope1.fetch()
    wfm2 = scope2.fetch()
```

configure_for_homogeneous_triggers

nitclk.**configure_for_homogeneous_triggers**(*sessions*)

Configures the properties commonly required for the TClk synchronization of device sessions with homogeneous triggers in a single PXI chassis or a single PC. Use `nitclk.configure_for_homogeneous_triggers()` to configure the properties for the reference clocks, start triggers, reference triggers, script triggers, and pause triggers. If `nitclk.configure_for_homogeneous_triggers()` cannot perform all the steps appropriate for the given sessions, it returns an error. If an error is returned, use the instrument driver methods and properties for signal routing, along with the following NI-TClk properties: `nitclk.SessionReference.start_trigger_master_session` *nitclk.SessionReference.ref_trigger_master_session* *nitclk.SessionReference.pause_trigger_master_session* *nitclk.configure_for_homogeneous_triggers()* affects the following clocks and triggers: - Reference clocks - Start triggers - Reference triggers - Script triggers - Pause triggers Reference Clocks *nitclk.configure_for_homogeneous_triggers()* configures the reference clocks if they are needed. Specifically, if the internal sample clocks or internal sample clock timebases are used, and the reference clock source is not configured–or is set to None (no trigger configured)–*nitclk.configure_for_homogeneous_triggers()* configures the following: PXI–The reference clock source on all devices is set to be the 10 MHz PXI backplane clock (PXI_CLK10). PCI–One of the devices exports its 10 MHz onboard reference clock to RTSI 7. The reference clock source on all devices is set to be RTSI 7. Note: If the reference clock source is set to a value other than None, *nitclk.configure_for_homogeneous_triggers()* cannot configure the reference clock source. Start Triggers If the start trigger is set to None (no trigger configured) for all sessions, the sessions are configured to share the start trigger. The start trigger is shared by: - Implicitly exporting the start trigger from one session - Configuring the other sessions for digital edge start triggers with sources corresponding to the exported start trigger - Setting *nitclk.SessionReference.start_trigger_master_session* to the session that is exporting the trigger for all sessions If the start triggers are None for all except one session, *nitclk.configure_for_homogeneous_triggers()* configures the sessions to share the start trigger from the one excepted session. The start trigger is shared by: - Implicitly exporting start trigger from the session with the start trigger that is not None - Configuring the other sessions for digital-edge start triggers with sources corresponding to the exported start trigger - Setting *nitclk.SessionReference.start_trigger_master_session* to the session that is exporting the trigger for all sessions If start triggers are configured for all sessions, *nitclk.configure_for_homogeneous_triggers()* does not affect the start triggers. Start triggers are considered to be configured for all sessions if either of the following conditions is true: - No session has a start trigger that is None - One session has a start trigger that is None, and all other sessions have start triggers other than None. The one session with the None trigger must have *nitclk.SessionReference.start_trigger_master_session* set to itself, indicating that the session itself is the start trigger master Reference Triggers *nitclk.configure_for_homogeneous_triggers()* configures sessions that support reference triggers to share the reference triggers if the reference triggers are None (no trigger configured) for all except one session. The reference triggers are shared by: - Implicitly exporting the reference trigger from the session whose reference trigger is not None - Configuring the other sessions that support the reference trigger for digital-edge reference triggers with sources corresponding to the exported reference trigger - Setting *nitclk.SessionReference.ref_trigger_master_session* to the session that is exporting the trigger for all sessions that support reference trigger If the reference triggers are configured for all sessions that support reference triggers, *nitclk.configure_for_homogeneous_triggers()* does not affect the reference triggers. Reference triggers are considered to be configured for all sessions if either one or the other of the following conditions is true: - No session has a reference trigger that is None - One session has a reference trigger that is None, and all other sessions have reference triggers other than None. The one session with the None trigger must have *nitclk.SessionReference.ref_trigger_master_session* set to itself, indicating that the session itself is the reference trigger master Reference Trigger Holdoffs Acquisition sessions may be configured with the reference trigger. For acquisition sessions, when the reference trigger is shared, *nitclk.configure_for_homogeneous_triggers()* configures the holdoff properties (which are instrument driver specific) on the reference trigger master session so that the session does not recognize the reference trigger before the other sessions are ready. This condition is only relevant when the sample clock rates, sample clock timebase rates, sample counts, holdoffs, and/or any delays for the acquisitions are different. When the sample clock rates, sample clock timebase rates, and/or the sample counts are different in acquisition sessions sharing the reference trigger, you should also set the holdoff properties for the reference trigger master using the instrument driver. Pause Triggers *nitclk.configure_for_homogeneous_triggers()* configures generation sessions that support pause triggers to share them, if the pause triggers are None (no trigger configured) for all except one session. The pause triggers are shared by: - Implicitly exporting the pause trigger from the session whose script trigger is not None - Configuring the other sessions that support the pause trigger for digital-edge pause triggers with sources corresponding to the exported pause trigger - Setting `nitclk.SessionReference.pause_trigger_master_session` to the session that is exporting the trigger for all sessions that support script triggers If the pause triggers are configured for all generation sessions that support pause triggers, `nitclk.configure_for_homogeneous_triggers()` does not affect pause triggers. Pause triggers are considered to be configured for all sessions if either one or the other of the following conditions is true: - No session has a pause trigger that is None - One session has a pause trigger that is None and all other sessions have pause triggers other than None. The one session with the None trigger must have `nitclk.SessionReference.pause_trigger_master_session` set to itself, indicating that the session itself is the pause trigger master Note: TClk synchronization is not supported for pause triggers on acquisition sessions.

Parameters **sessions** (*list of instrument-specific sessions or nitclk.SessionReference instances*) – sessions is an array of sessions that are being synchronized.

finish_sync_pulse_sender_synchronize

`nitclk.finish_sync_pulse_sender_synchronize`(*sessions*, *min_time=hightime.timedelta(seconds=0.0)*)

Finishes synchronizing the Sync Pulse Sender.

Parameters

- **sessions** (*list of instrument-specific sessions or nitclk.SessionReference instances*) – sessions is an array of sessions that are being synchronized.
- **min_time** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Minimal period of TClk, expressed in seconds. Supported values are between 0.0 s and 0.050 s (50 ms). Minimal period for a single chassis/PC is 200 ns. If the specified value is less than 200 ns, NI-TClk automatically coerces minTime to 200 ns. For multichassis synchronization, adjust this value to account for propagation delays through the various devices and cables.

initiate

`nitclk.initiate`(*sessions*)

Initiates the acquisition or generation sessions specified, taking into consideration any special requirements needed for synchronization. For example, the session exporting the TClk-synchronized start trigger is not initiated until after `nitclk.initiate()` initiates all the sessions that import the TClk-synchronized start trigger.

Parameters **sessions** (*list of instrument-specific sessions or nitclk.SessionReference instances*) – sessions is an array of sessions that are being synchronized.

is_done

`nitclk.is_done`(*sessions*)

Monitors the progress of the acquisitions and/or generations corresponding to sessions.

Parameters **sessions** (*list of instrument-specific sessions or nitclk.SessionReference instances*) – sessions is an array of sessions that are being synchronized.

Return type bool

Returns Indicates that the operation is done. The operation is done when each session has completed without any errors or when any one of the sessions reports an error.

setup_for_sync_pulse_sender_synchronize

nitclk.**setup_for_sync_pulse_sender_synchronize**(*sessions*, *min_time=hightime.timedelta(seconds=0.0)*)

Configures the TClks on all the devices and prepares the Sync Pulse Sender for synchronization

Parameters

- **sessions** (*list of instrument-specific sessions or nitclk.SessionReference instances*) – sessions is an array of sessions that are being synchronized.
- **min_time** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Minimal period of TClk, expressed in seconds. Supported values are between 0.0 s and 0.050 s (50 ms). Minimal period for a single chassis/PC is 200 ns. If the specified value is less than 200 ns, NI-TClk automatically coerces minTime to 200 ns. For multichassis synchronization, adjust this value to account for propagation delays through the various devices and cables.

synchronize

nitclk.**synchronize**(*sessions*, *min_tclk_period=hightime.timedelta(seconds=0.0)*)

Synchronizes the TClk signals on the given sessions. After `nitclk.synchronize()` executes, TClk signals from all sessions are synchronized. Note: Before using this NI-TClk method, verify that your system is configured as specified in the PXI Trigger Lines and RTSI Lines topic of the NI-TClk Synchronization Help. You can locate this help file at Start>>Programs>>National Instruments>>NI-TClk.

Parameters

- **sessions** (*list of instrument-specific sessions or nitclk.SessionReference instances*) – sessions is an array of sessions that are being synchronized.
- **min_tclk_period** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Minimal period of TClk, expressed in seconds. Supported values are between 0.0 s and 0.050 s (50 ms). Minimal period for a single chassis/PC is 200 ns. If the specified value is less than 200 ns, NI-TClk automatically coerces minTime to 200 ns. For multichassis synchronization, adjust this value to account for propagation delays through the various devices and cables.

synchronize_to_sync_pulse_sender

nitclk.**synchronize_to_sync_pulse_sender**(*sessions*, *min_time=hightime.timedelta(seconds=0.0)*)

Synchronizes the other devices to the Sync Pulse Sender.

Parameters

- **sessions** (*list of instrument-specific sessions or nitclk.SessionReference instances*) – sessions is an array of sessions that are being synchronized.
- **min_time** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Minimal period of TClk, expressed in seconds. Supported values are between 0.0 s and 0.050 s (50 ms). Minimal period for a single chassis/PC is 200 ns. If the specified value is less than 200 ns, NI-TClk automatically coerces minTime to 200 ns. For multichassis synchronization, adjust this value to account for propagation delays through the various devices and cables.

wait_until_done

nitclk.**wait_until_done**(*sessions, timeout=hightime.timedelta(seconds=0.0)*)

Call this method to pause execution of your program until the acquisitions and/or generations corresponding to sessions are done or until the method returns a timeout error. *nitclk.wait_until_done()* is a blocking method that periodically checks the operation status. It returns control to the calling program if the operation completes successfully or an error occurs (including a timeout error). This method is most useful for finite data operations that you expect to complete within a certain time.

Parameters

- **sessions** (*list of instrument-specific sessions or nitclk.SessionReference instances*) – sessions is an array of sessions that are being synchronized.
- **timeout** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The amount of time in seconds that *nitclk.wait_until_done()* waits for the sessions to complete. If timeout is exceeded, *nitclk.wait_until_done()* returns an error.

SessionReference

class nitclk.**SessionReference**(*session_number*)

Helper class that contains all NI-TClk properties. This class is what is returned by any nimi-python Session class tclk attribute when the driver supports NI-TClk

```
with niscope.Session('dev1') as session:
    session.tclk.sample_clock_delay = .42
```

..note:: Constructing this class is an advanced use case and should not be needed in most circumstances.

Parameters **session_number** (*int, nimi-python Session class, SessionReference*) – nitclk session

exported_sync_pulse_output_terminal

nitclk.SessionReference.**exported_sync_pulse_output_terminal**

Specifies the destination of the Sync Pulse. This property is most often used when synchronizing a multichassis system. Values Empty string. Empty string is a valid value, indicating that the signal is not exported. PXI Devices - 'PXI_Trig0' through 'PXI_Trig7' and device-specific settings PCI Devices - 'RTSI_0' through 'RTSI_7' and device-specific settings Examples of Device-Specific

Settings - NI PXI-5122 supports 'PFI0' and 'PFI1' - NI PXI-5421 supports 'PFI0', 'PFI1', 'PFI4', and 'PFI5' - NI PXI-6551/6552 supports 'PFI0', 'PFI1', 'PFI2', and 'PFI3' Default Value is empty string

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Export Sync Pulse Output Terminal**
- C Attribute: **NITCLK_ATTR_EXPORTED_SYNC_PULSE_OUTPUT_TERMINAL**

exported_tclk_output_terminal

`nitclk.SessionReference.`**`exported_tclk_output_terminal`**

Specifies the destination of the device's TClk signal. Values Empty string. Empty string is a valid value, indicating that the signal is not exported. PXI Devices - 'PXI_Trig0' through 'PXI_Trig7' and device-specific settings PCI Devices - 'RTSI_0' through 'RTSI_7' and device-specific settings Examples of Device-Specific Settings - NI PXI-5122 supports 'PFI0' and 'PFI1' - NI PXI-5421 supports 'PFI0', 'PFI1', 'PFI4', and 'PFI5' - NI PXI-6551/6552 supports 'PFI0', 'PFI1', 'PFI2', and 'PFI3' Default Value is empty string

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Output Terminal**
- C Attribute: **NITCLK_ATTR_EXPORTED_TCLK_OUTPUT_TERMINAL**

pause_trigger_master_session

`nitclk.SessionReference.`**`pause_trigger_master_session`**

Specifies the pause trigger master session. For external triggers, the session that originally receives the trigger. For None (no trigger configured) or software triggers, the session that originally generates the trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	instrument-specific session or an instance of nitclk.SessionReference
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Pause Trigger Master Session**
- C Attribute: **NITCLK_ATTR_PAUSE_TRIGGER_MASTER_SESSION**

ref_trigger_master_session

`nitclk.SessionReference.`**`ref_trigger_master_session`**

Specifies the reference trigger master session. For external triggers, the session that originally receives the trigger. For None (no trigger configured) or software triggers, the session that originally generates the trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	instrument-specific session or an instance of nitclk.SessionReference
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Reference Trigger Master Session**
- C Attribute: **NITCLK_ATTR_REF_TRIGGER_MASTER_SESSION**

sample_clock_delay

`nitclk.SessionReference.`**`sample_clock_delay`**

Specifies the sample clock delay. Specifies the delay, in seconds, to apply to the session sample clock relative to the other synchronized sessions. During synchronization, NI-TClk aligns the sample clocks on the synchronized devices. If you want to delay the sample clocks, set this property before calling *`nitclk.synchronize()`*. not supported for acquisition sessions. Values - Between minus one and plus one period of the sample clock. One sample clock period is equal to (1/sample clock rate). For example, for a session with sample rate of 100 MS/s, you can specify sample clock delays between -10.0 ns and +10.0 ns. Default Value is 0

Note: Sample clock delay is supported for generation sessions only; it is

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	hightime.timedelta, datetime.timedelta, or float in seconds
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Sample Clock Delay**
- C Attribute: **NITCLK_ATTR_SAMPLE_CLOCK_DELAY**

sequencer_flag_master_session

`nitclk.SessionReference.`**`sequencer_flag_master_session`**

Specifies the sequencer flag master session. For external triggers, the session that originally receives the trigger. For None (no trigger configured) or software triggers, the session that originally generates the trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	instrument-specific session or an instance of nitclk.SessionReference
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Sequencer Flag Master Session**
- C Attribute: **NITCLK_ATTR_SEQUENCER_FLAG_MASTER_SESSION**

start_trigger_master_session

`nitclk.SessionReference.`**`start_trigger_master_session`**

Specifies the start trigger master session. For external triggers, the session that originally receives the trigger. For None (no trigger configured) or software triggers, the session that originally generates the trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	instrument-specific session or an instance of nitclk.SessionReference
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Start Trigger Master Session**
- C Attribute: **NITCLK_ATTR_START_TRIGGER_MASTER_SESSION**

sync_pulse_clock_source

`nitclk.SessionReference.`**`sync_pulse_clock_source`**

Specifies the Sync Pulse Clock source. This property is typically used to synchronize PCI devices when you want to control RTSI 7 yourself. Make sure that a 10 MHz clock is driven onto RTSI 7. Values PCI Devices - 'RTSI_7' and 'None' PXI Devices - 'PXI_CLK10' and 'None' Default Value - 'None' directs *`nitclk.synchronize()`* to create the necessary routes. For PCI, one of the synchronized devices drives a 10 MHz clock on RTSI 7 unless that line is already being driven.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Sync Pulse Clock Source**
- C Attribute: **NITCLK_ATTR_SYNC_PULSE_CLOCK_SOURCE**

sync_pulse_sender_sync_pulse_source

`nitclk.SessionReference.`**`sync_pulse_sender_sync_pulse_source`**

Specifies the external sync pulse source for the Sync Pulse Sender. You can use this source to synchronize the Sync Pulse Sender with an external non-TClk source. Values Empty string. Empty string is a valid value, indicating that the signal is not exported. PXI Devices - 'PXI_Trig0' through 'PXI_Trig7' and device-specific settings PCI Devices - 'RTSI_0' through 'RTSI_7' and device-specific settings Examples of Device-Specific Settings - NI PXI-5122 supports 'PFI0' and 'PFI1' - NI PXI-5421 supports 'PFI0', 'PFI1', 'PFI4', and 'PFI5' - NI PXI-6551/6552 supports 'PFI0', 'PFI1', 'PFI2', and 'PFI3' Default Value is empty string

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **External Pulse Source**
- C Attribute: **NITCLK_ATTR_SYNC_PULSE_SENDER_SYNC_PULSE_SOURCE**

sync_pulse_source

`nitclk.SessionReference.`**`sync_pulse_source`**

Specifies the Sync Pulse source. This property is most often used when synchronizing a multichassis system. Values Empty string PXI Devices - 'PXI_Trig0' through 'PXI_Trig7' and device-specific settings PCI Devices - 'RTSI_0' through 'RTSI_7' and device-specific settings Examples of Device-Specific Settings - NI PXI-5122 supports 'PFI0' and 'PFI1' - NI PXI-5421 supports 'PFI0', 'PFI1', 'PFI2', and 'PFI3' - NI PXI-6551/6552 supports 'PFI0', 'PFI1', 'PFI2', and 'PFI3' Default Value - Empty string. This default value directs `nitclk.synchronize()` to set this property when all the synchronized devices are in one PXI chassis. To synchronize a multichassis system, you must set this property before calling `nitclk.synchronize()`.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Sync Pulse Source**
- C Attribute: **NITCLK_ATTR_SYNC_PULSE_SOURCE**

tclk_actual_period

`nitclk.SessionReference.`**`tclk_actual_period`**

Indicates the computed TClk period that will be used during the acquisition.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read only

Tip: This property corresponds to the following LabVIEW Property or C Attribute:

- LabVIEW Property: **Period**
- C Attribute: **NITCLK_ATTR_TCLK_ACTUAL_PERIOD**

nitclk

Exceptions and Warnings

Error

exception `nitclk.errors.`**`Error`**

Base exception type that all NI-TClk exceptions derive from

DriverError

exception `nitclk.errors.`**`DriverError`**

An error originating from the NI-TClk driver

UnsupportedConfigurationError

exception `nitclk.errors.`**`UnsupportedConfigurationError`**

An error due to using this module in an usupported platform.

DriverNotInstalledError

exception `nitclk.errors.`**`DriverNotInstalledError`**

An error due to using this module without the driver runtime installed.

DriverWarning

exception nitclk.errors.**DriverWarning**
A warning originating from the NI-TClk driver

Examples

You can download all nitclk examples here

nitclk_niscope_synchronize_with_trigger.py

Listing 22: (nitclk_niscope_synchronize_with_trigger.py)

```python
import argparse
import niscope
import nitclk
import sys
import time


def example(resource_name1, resource_name2, options):
    with niscope.Session(resource_name=resource_name1, options=options) as session1, niscope.Session(resource_name=resource_name2, options=options) as session2:
        session_list = [session1, session2]
        for session in session_list:
            session.configure_vertical(range=1.0, coupling=niscope.VerticalCoupling.DC)
            session.configure_horizontal_timing(min_sample_rate=50000000, min_num_pts=1000, ref_position=50.0, num_records=1, enforce_realtime=True)
        session1.trigger_type = niscope.TriggerType.SOFTWARE
        nitclk.configure_for_homogeneous_triggers(session_list)
        nitclk.synchronize(session_list, 200e-9)
        nitclk.initiate(session_list)
        time.sleep(100)
        session1.send_software_trigger_edge(niscope.WhichTrigger.START)
        waveforms = session2.channels[0].fetch(num_samples=1000)
        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='Synchronizes multiple instruments to one trigger.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n1', '--resource-name1', default='PXI1Slot2', help='Resource name of a NI Digitizer')
    parser.add_argument('-n2', '--resource-name2', default='PXI1Slot3', help='Resource name of a NI Digitizer')
    parser.add_argument('-op', '--option-string', default='', type=str, help='Option string')
    args = parser.parse_args(argsv)
    example(args.resource_name1, args.resource_name2, args.option_string)


```

(continues on next page)

(continued from previous page)

```
def main():
    _main(sys.argv[1:])


def test_example():
    options = {'simulate': True, 'driver_setup': {'Model': '5164', 'BoardType': 'PXIe
↪', }, }
    example('PXI1Slot2', 'PXI1Slot13', options)


def test_main():
    cmd_line = ['--option-string', 'Simulate=1, DriverSetup=Model:5164; BoardType:PXIe
↪', ]
    _main(cmd_line)


if __name__ == '__main__':
    main()
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index