NI-SWITCH Python API Documentation

Release 1.4.8

NI

Apr 26, 2024

DOCUMENTATION

CHAPTER

ONE

ABOUT

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

1.1 Support Policy

niswitch supports all the Operating Systems supported by NI-SWITCH.

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

CHAPTER

TWO

CONTRIBUTING

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

CHAPTER

THREE

SUPPORT / FEEDBACK

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

CHAPTER

FOUR

BUGS / FEATURE REQUESTS

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

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

4.1 niswitch module

4.1.1 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~=1.4.8
```

4.1.2 Usage

The following is a basic example of using the **niswitch** module to open a session to a Switch and connect channels.

```
import niswitch
with niswitch.Session("Dev1") as session:
    session.connect(channel1='r0', channel2='c0')
```

Other usage examples can be found on GitHub.

4.1.3 API Reference

Session

class niswitch.**Session**(*self*, *resource_name*, *topology='Configured Topology'*, *simulate=False*, *reset_device=False*, ***, *grpc_options=None*)

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 topology created when you configured the device in MAX, using either "Configured Topology" or the topology string of the device. Refer to the Initializing with Topology 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 "Configured Topology" to use the last topology that was configured for the device in MAX. Default Value: "Configured Topology" Valid Values: "Configured Topology" "2501/1-Wire 48x1 Mux" "2501/1-Wire 48x1 Amplified Mux" "2501/2-Wire 24x1 Mux" "2501/2-Wire 24x1 Amplified Mux" "2501/2-Wire Dual 12x1 Mux" "2501/2-Wire Quad 6x1 Mux" "2501/2-Wire 4x6 Matrix" "2501/4-Wire 12x1 Mux" "2503/1-Wire 48x1 Mux" "2503/2-Wire 24x1 Mux" "2503/2-Wire Dual 12x1 Mux" "2503/2-Wire Quad 6x1 Mux" "2503/2-Wire 4x6 Matrix" "2503/4-Wire 12x1 Mux" "2510/Independent" "2512/Independent" "2514/Independent" "2515/Independent" "2520/80-SPST" "2521/40-DPST" "2522/53-SPDT" "2523/26-DPDT" "2524/1-Wire 128x1 Mux" "2524/1-Wire Dual 64x1 Mux" "2524/1-Wire Quad 32x1 Mux" "2524/1-Wire Octal 16x1 Mux" "2524/1-Wire Sixteen 8x1 Mux" "2525/2-Wire 64x1 Mux" "2525/2-Wire Dual 32x1 Mux" "2525/2-Wire Quad 16x1 Mux" "2525/2-Wire Octal 8x1 Mux" "2525/2-Wire Sixteen 4x1 Mux" "2526/1-Wire 158x1 Mux" "2526/2-Wire 79x1 Mux" "2527/1-Wire 64x1 Mux" "2527/1-Wire Dual 32x1 Mux" "2527/2-Wire 32x1 Mux" "2527/2-Wire Dual 16x1 Mux" "2527/4-Wire 16x1 Mux" "2527/Independent" "2529/2-Wire Dual 4x16 Matrix" "2529/2-Wire 8x16 Matrix" "2529/2-Wire 4x32 Matrix" "2530/1-Wire 128x1 Mux" "2530/1-Wire Dual 64x1 Mux" "2530/1-Wire 4x32 Matrix" "2530/1-Wire 8x16 Matrix" "2530/1-Wire Octal 16x1 Mux" "2530/1-Wire Quad 32x1 Mux" "2530/2-Wire 4x16 Matrix" "2530/2-Wire 64x1 Mux" "2530/2-Wire Dual 32x1 Mux" "2530/2-Wire Quad 16x1 Mux" "2530/4-Wire 32x1 Mux" "2530/4-Wire Dual 16x1 Mux" "2530/Independent" "2531/1-Wire 4x128 Matrix" "2531/1-Wire 8x64 Matrix" "2531/1-Wire Dual 4x64 Matrix" "2531/1-Wire Dual 8x32 Matrix" "2531/2-Wire 4x64 Matrix" "2531/2-Wire 8x32 Matrix" "2532/1-Wire 16x32 Matrix" "2532/1-Wire 4x128 Matrix" "2532/1-Wire 8x64 Matrix" "2532/1-Wire Dual 16x16 Matrix" "2532/1-Wire Dual 4x64 Matrix" "2532/1-Wire Dual 8x32 Matrix" "2532/1-Wire Quad 4x32 Matrix" "2532/1-Wire Sixteen 2x16 Matrix" "2532/2-Wire 16x16 Matrix" "2532/2-Wire 4x64 Matrix" "2532/2-Wire 8x32 Matrix" "2532/2-Wire Dual 4x32 Matrix" "2533/1-Wire 4x64 Matrix" "2534/1-Wire 8x32 Matrix" "2535/1-Wire 4x136 Matrix" "2536/1-Wire 8x68 Matrix" "2540/1-Wire 8x9 Matrix" "2541/1-Wire 8x12 Matrix" "2542/Quad 2x1 Terminated Mux" "2543/Dual 4x1 Terminated Mux" "2544/8x1 Terminated Mux" "2545/4x1 Terminated Mux" "2546/Dual 4x1 Mux" "2547/8x1 Mux" "2548/4-SPDT" "2549/Terminated 2-SPDT" "2554/4x1 Mux" "2555/4x1 Terminated Mux" "2556/Dual 4x1 Mux" "2557/8x1 Mux" "2558/4-SPDT" "2559/Terminated 2-SPDT" "2564/16-SPST" "2564/8-DPST" "2565/16-SPST" "2566/16-SPDT" "2566/8-DPDT" "2567/Independent" "2568/15-DPST" "2568/31-SPST" "2569/100-SPST" "2569/50-DPST" "2570/20-DPDT" "2570/40-SPDT" "2571/66-SPDT" "2575/1-Wire 196x1 Mux" "2575/2-Wire 98x1 Mux" "2575/2-Wire 95x1 Mux" "2576/2-Wire 64x1 Mux" "2576/2-Wire Dual 32x1 Mux" "2576/2-Wire Octal 8x1 Mux" "2576/2-Wire Quad 16x1 Mux" "2576/2-Wire Sixteen 4x1 Mux"

"2576/Independent" "2584/1-Wire 12x1 Mux" "2584/1-Wire Dual 6x1 Mux" "2584/2-Wire 6x1 Mux" "2584/Independent" "2585/1-Wire 10x1 Mux" "2586/10-SPST" "2586/5-DPST" "2590/4x1 Mux" "2591/4x1 Mux" "2593/16x1 Mux" "2593/8x1 Terminated Mux" "2593/Dual 8x1 Mux" "2593/Dual 4x1 Terminated Mux" "2593/Independent" "2594/4x1 Mux" "2595/4x1 Mux" "2596/Dual 6x1 Mux" "2597/6x1 Terminated Mux" "2598/Dual Transfer" "2599/2-SPDT" "2720/Independent" "2722/Independent" "2725/Independent" "2727/Independent" "2737/2-Wire 4x64 Matrix" "2738/2-Wire 8x32 Matrix" "2739/2-Wire 16x16 Matrix" "2746/Quad 4x1 Mux" "2747/Dual 8x1 Mux" "2748/16x1 Mux" "2790/Independent" "2796/Dual 6x1 Mux" "2797/6x1 Terminated Mux" "2798/Dual Transfer" "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.
- **grpc_options** (niswitch.GrpcSessionOptions) – MeasurementLink gRPC session options

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

```
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

- *Session*
- *Methods*
  - *abort*

- *handshaking_initiation*
- *instrument_firmware_revision*
- *instrument_manufacturer*
- *instrument_model*
- *io_resource_descriptor*
- *is_configuration_channel*
- *is_debounced*
- *is_scanning*
- *is_source_channel*
- *is_waiting_for_trig*
- *logical_name*
- *max_ac_voltage*
- *max_carry_ac_current*
- *max_carry_ac_power*
- *max_carry_dc_current*
- *max_carry_dc_power*
- *max_dc_voltage*
- *max_switching_ac_current*
- *max_switching_ac_power*
- *max_switching_dc_current*
- *max_switching_dc_power*
- *number_of_relays*
- *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.

DriverTooOldError

exception niswitch.errors.**DriverTooOldError**

An error due to using this module with an older version of the NI-SWITCH driver runtime.

DriverTooNewError

exception niswitch.errors.**DriverTooNewError**

An error due to the NI-SWITCH driver runtime being too new for this module.

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

RpcError

exception niswitch.errors.**RpcError**

An error specific to sessions to the NI gRPC Device Server

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 1: (niswitch_connect_channels.py)

```
#!/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 an NI switch.')
    parser.add_argument('-ch1', '--channel1', default='c0', help='Channel One.')
```

(continues on next page)

(continued from previous page)

```
    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 2: (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))
```

(continues on next page)

(continued from previous page)

```
            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 NI-SWITCH.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource name of an NI 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.')
    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)

```

(continues on next page)

(continued from previous page)

```
59
60 def test_main():
61     cmd_line = ['--topology', '2737/2-Wire 4x64 Matrix', '--simulate', '--device', '--
   ↪channel', '--relay', ]
62     _main(cmd_line)
63
64
65 def main():
66     _main(sys.argv[1:])
67
68
69 if __name__ == '__main__':
70     main()
71
72
```

niswitch_relay_control.py

Listing 3: (niswitch_relay_control.py)

```
1  #!/usr/bin/python
2
3  import argparse
4  import niswitch
5  import sys
6
7
8  def example(resource_name, topology, simulate, relay, action):
9      # if we are simulating resource name must be blank
10     resource_name = '' if simulate else resource_name
11
12     with niswitch.Session(resource_name=resource_name, topology=topology,␣
   ↪simulate=simulate) as session:
13         session.relay_control(relay_name=relay, relay_action=niswitch.
   ↪RelayAction[action])
14         print('Relay ', relay, ' has had the action ', action, ' performed.')
15
16
17 def _main(argsv):
18     parser = argparse.ArgumentParser(description='Performs relay control with NI-SWITCH␣
   ↪relays.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
19     parser.add_argument('-n', '--resource-name', default='PXI1Slot2', help='Resource␣
   ↪name of an NI switch.')
20     parser.add_argument('-r', '--relay', default='k0', help='Relay Name.')
21     parser.add_argument('-a', '--action', default='OPEN', choices=niswitch.RelayAction.__
   ↪members__.keys(), type=str.upper, help='Relay Action.')
22     parser.add_argument('-t', '--topology', default='Configured Topology', help=
   ↪'Topology.')
23     parser.add_argument('-s', '--simulate', default=False, action='store_true', help=
   ↪'Simulate device.')
```

(continues on next page)

(continued from previous page)

```
    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()


```

gRPC Support

Support for using NI-SWITCH over gRPC

SessionInitializationBehavior

class niswitch.**SessionInitializationBehavior**

AUTO

The NI gRPC Device Server will attach to an existing session with the specified name if it exists, otherwise the server will initialize a new session.

Note: When using the Session as a context manager and the context exits, the behavior depends on what happened when the constructor was called. If it resulted in a new session being initialized on the NI gRPC Device Server, then it will automatically close the server session. If it instead attached to an existing session, then it will detach from the server session and leave it open.

INITIALIZE_SERVER_SESSION

Require the NI gRPC Device Server to initialize a new session with the specified name.

Note: When using the Session as a context manager and the context exits, it will automatically close the server session.

ATTACH_TO_SERVER_SESSION

Require the NI gRPC Device Server to attach to an existing session with the specified name.

Note: When using the Session as a context manager and the context exits, it will detach from the server session and leave it open.

GrpcSessionOptions

class niswitch.**GrpcSessionOptions**(*self*, *grpc_channel*, *session_name*, *initialization_behavior=SessionInitializationBehavior.AUTO*)

Collection of options that specifies session behaviors related to gRPC.

Creates and returns an object you can pass to a Session constructor.

Parameters

- **grpc_channel** (*grpc.Channel*) – Specifies the channel to the NI gRPC Device Server.
- **session_name** (*str*) – User-specified name that identifies the driver session on the NI gRPC Device Server.

  This is different from the resource name parameter many APIs take as a separate parameter. Specifying a name makes it easy to share sessions across multiple gRPC clients. You can use an empty string if you want to always initialize a new session on the server. To attach to an existing session, you must specify the session name it was initialized with.
- **initialization_behavior** (*niswitch.SessionInitializationBehavior*) – Specifies whether it is acceptable to initialize a new session or attach to an existing one, or if only one of the behaviors is desired.

  The driver session exists on the NI gRPC Device Server.

4.2 Additional Documentation

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

Refer to the nimi-python Read the Docs project for documentation of versions 1.4.4 of the module or earlier.

CHAPTER

FIVE

LICENSE

nimi-python is licensed under an MIT-style license (see LICENSE). Other incorporated projects may be licensed under different licenses. All licenses allow for non-commercial and commercial use.

gRPC Features

For driver APIs that support it, passing a GrpcSessionOptions instance as a parameter to Session.__init__() is subject to the NI General Purpose EULA (see NILICENSE).

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

G

H

I

L

M

N

O

P

R

S

T

U

W