NI Switch Executive Python API Documentation

Release 1.4.9.dev0

NI

May 01, 2024

DOCUMENTATION

CHAPTER

ONE

ABOUT

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

1.1 Support Policy

nise supports all the Operating Systems supported by NI Switch Executive.

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 nise module

4.1.1 Installation

As a prerequisite to using the **nise** module, you must install the NI Switch Executive runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

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

```
$ python -m pip install nise
```

4.1.2 Usage

The following is a basic example of using the **nise** module to open a session to a Switch Executive Virtual Device and connect a routegroup.

```
import nise
with nise.Session('SwitchExecutiveExample') as session:
    session.connect('DIOToUUT')
```

Other usage examples can be found on GitHub.

4.1.3 API Reference

Session

class nise.**Session**(*self*, *virtual_device_name*, *options={}*)

Opens a session to a specified NI Switch Executive virtual device. Opens communications with all of the IVI switches associated with the specified NI Switch Executive virtual device. Returns a session handle that you use to identify the virtual device in all subsequent NI Switch Executive method calls. NI Switch Executive uses a reference counting scheme to manage open session handles to an NI Switch Executive virtual device. Each call to `nise.Session.__init__()` must be matched with a subsequent call to *`nise.Session.close()`*. Successive calls to `nise.Session.__init__()` with the same virtual device name always returns the same session handle. NI Switch Executive disconnects its communication with the IVI switches after all session handles are closed to

a given virtual device. The session handles may be used safely in multiple threads of an application. Sessions may only be opened to a given NI Switch Executive virtual device from a single process at a time.

Parameters

- **virtual_device_name** (*str*) – The name of the NI Switch Executive virtual device.
- **options** (*dict*) – Specifies the initial value of certain properties for the session. The syntax for **options** is a dictionary of properties with an assigned value. For example:

  { 'simulate': False }

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

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

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

Methods

close

nise.Session.**close**()

Reduces the reference count of open sessions by one. If the reference count goes to 0, the method deallocates any memory resources the driver uses and closes any open IVI switch sessions. After calling the *nise.Session.close()* method, you should not use the NI Switch Executive virtual device again until you call nise.Session.__init__().

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

connect

nise.Session.**connect**(*connect_spec*, *multiconnect_mode=nise.MulticonnectMode.DEFAULT*, *wait_for_debounce=True*)

Connects the routes specified by the connection specification. When connecting, it may allow for multiconnection based on the multiconnection mode. In the event of an error, the call to *nise.Session.connect()* will attempt to undo any connections made so that the system will be left in the same state that it was in before the call was made. Some errors can be caught before manipulating hardware, although it is feasible that a hardware call could fail causing some connections to be momentarily closed and then reopened. If the wait for debounce parameter is set, the method will not return until the switch system has debounced.

Parameters

- **connect_spec** (*str*) – String describing the connections to be made. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.
- **multiconnect_mode** (*nise.MulticonnectMode*) – This value sets the connection mode for the method. The mode might be one of the following: NISE_VAL_USE_DEFAULT_MODE (-1) - uses the mode selected as the default for the route in the NI Switch Executive virtual device configuration. If a mode has not been selected for the route in the NI Switch Executive virtual device, this parameter defaults to NISE_VAL_MULTICONNECT_ROUTES. *NO_MULTICONNECT* (0) - routes specified in the connection specification must be disconnected before they can be reconnected. Calling Connect on a route that was connected using No Multiconnect mode results in an error condition. NISE_VAL_MULTICONNECT_ROUTES (1)- routes specified in the connection specification can be connected multiple times. The first call to Connect performs the physical hardware connection. Successive calls to Connect increase a connection reference count. Similarly, calls to Disconnect decrease the reference count. Once it reaches 0, the hardware is physically disconnected. Multiconnecting routes applies to entire routes and not to route segments.

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

- **wait_for_debounce** (*bool*) – Waits (if true) for switches to debounce between its connect and disconnect operations. If false, it immediately begins the second operation after completing the first. The order of connect and disconnect operation is set by the Operation Order input.

connect_and_disconnect

nise.Session.**connect_and_disconnect**(*connect_spec*, *disconnect_spec*, *multiconnect_mode=nise.MulticonnectMode.DEFAULT*, *operation_order=nise.OperationOrder.AFTER*, *wait_for_debounce=True*)

Connects routes and disconnects routes in a similar fashion to *nise.Session.connect()* and *nise.Session.disconnect()* except that the operations happen in the context of a single method call. This method is useful for switching from one state to another state. *nise.Session.connect_and_disconnect()* manipulates the hardware connections and disconnections only when the routes are different between the connection and disconnection specifications. If any routes are common between the connection and disconnection specifications, NI Switch Executive determines whether or not the relays need to be switched. This functionality has the distinct advantage of increased throughput for shared connections, because hardware does not have to be involved and potentially increases relay lifetime by decreasing the number of times that the relay has to be switched. In the event of an error, the call to *nise.Session.connect_and_disconnect()* attempts to undo any connections made, but does not attempt to reconnect disconnections. Some errors can be caught before manipulating hardware, although it is feasible that a hardware call could fail causing some connections to be momentarily closed and then reopened.

Parameters

- **connect_spec** (*str*) – String describing the connections to be made. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.
- **disconnect_spec** (*str*) – String describing the disconnections to be made. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.
- **multiconnect_mode** (*nise.MulticonnectMode*) – This value sets the connection mode for the method. The mode might be one of the following: NISE_VAL_USE_DEFAULT_MODE (-1) - uses the mode selected as the default for the route in the NI Switch Executive virtual device configuration. If a mode has not been selected for the route in the NI Switch Executive virtual device, this parameter defaults to NISE_VAL_MULTICONNECT_ROUTES. *NO_MULTICONNECT* (0) - routes specified in the connection specification must be disconnected before they can be reconnected. Calling Connect on a route that was connected using No Multiconnect mode results in an error condition. NISE_VAL_MULTICONNECT_ROUTES (1) - routes specified in the connection specification can be connected multiple times. The first call to Connect performs the physical hardware connection. Successive calls to Connect increase a connection reference count. Similarly, calls to Disconnect decrease the reference count. Once it reaches 0, the hardware is physically disconnected. This behavior is slightly different with SPDT relays. For more information, refer to the Exclusions and SPDT Relays topic in the NI Switch Executive Help. Multiconnecting routes applies to entire routes and not to route segments.

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

- **operation_order** (*nise.OperationOrder*) – Sets the order of the operation for the method. Defined values are Break Before Make and Break After Make. *BEFORE* (1) - The method disconnects the routes specified in the disconnect specification before connecting the routes specified in the connect specification. This is the typical mode of operation. *AFTER* (2) - The method connects the routes specified in the connection specification before connecting the routes specified in the disconnection specification. This mode of operation is normally used when you are switching current and want to ensure that a load is always connected to your source. The order of operation is to connect first or disconnect first.
- **wait_for_debounce** (*bool*) – Waits (if true) for switches to debounce between its connect and disconnect operations. If false, it immediately begins the second operation after completing the first. The order of connect and disconnect operation is set by the Operation Order input.

disconnect

nise.Session.**disconnect**(*disconnect_spec*)

Disconnects the routes specified in the Disconnection Specification. If any of the specified routes were originally connected in a multiconnected mode, the call to *nise.Session.disconnect()* reduces the reference count on the route by 1. If the reference count reaches 0, it is disconnected. If a specified route does not exist, it is an error condition. In the event of an error, the call to *nise.Session.disconnect()* continues to try to disconnect everything specified by the route specification string but reports the error on completion.

Parameters

disconnect_spec (*str*) – String describing the disconnections to be made. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.

disconnect_all

nise.Session.**disconnect_all**()

Disconnects all connections on every IVI switch device managed by the NISE session reference passed to this method. *nise.Session.disconnect_all()* ignores all multiconnect modes. Calling *nise.Session.disconnect_all()* resets all of the switch states for the system.

expand_route_spec

nise.Session.**expand_route_spec**(*route_spec*, *expand_action=nise.ExpandAction.ROUTES*, *expanded_route_spec_size=[1024]*)

Expands a route spec string to yield more information about the routes and route groups within the spec. The route specification string returned from *nise.Session.expand_route_spec()* can be passed to other Switch Executive API methods (such as *nise.Session.connect()*, *nise.Session.disconnect()*, and *nise.Session.connect_and_disconnect()*) that use route specification strings.

Parameters

- **route_spec** (*str*) – String describing the routes and route groups to expand. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.
- **expand_action** (*nise.ExpandAction*) – This value sets the expand action for the method. The action might be one of the following: *ROUTES* (0) - expands the route spec to routes. Converts route groups to their constituent routes. *PATHS* (1) - expands the route spec to paths. Converts routes and route groups to their constituent square bracket route spec strings. Example: [Dev1/c0->Dev1/r0->Dev1/c1]

- **expanded_route_spec_size** (*list of int*) – The routeSpecSize is an ViInt32 that is passed by reference into the method. As an input, it is the size of the route spec string buffer being passed. If the route spec string is larger than the string buffer being passed, only the portion of the route spec string that can fit in the string buffer is copied into it. On return from the method, routeSpecSize holds the size required to hold the entire route spec string. Note that this size may be larger than the buffer size as the method always returns the size needed to hold the entire buffer. You may pass NULL for this parameter if you are not interested in the return value for routeSpecSize and routeSpec.

Return type

str

Returns

The expanded route spec. Route specification strings can be directly passed to *nise.Session.connect()*, *nise.Session.disconnect()*, or *nise.Session.connect_and_disconnect()* Refer to Route Specification Strings in the NI Switch Executive Help for more information. You may pass NULL for this parameter if you are not interested in the return value. To obtain the route specification string, you should pass a buffer to this parameter. The size of the buffer required may be obtained by calling the method with NULL for this parameter and a valid ViInt32 to routeSpecSize. The routeSpecSize will contain the size needed to hold the entire route specification (including the NULL termination character). Common operation is to call the method twice. The first time you call the method you can determine the size needed to hold the route specification string. Allocate a buffer of the appropriate size and then re-call the method to obtain the entire buffer.

find_route

nise.Session.**find_route**(*channel1*, *channel2*, *route_spec_size=[1024]*)

Finds an existing or potential route between channel 1 and channel 2. The returned route specification contains the route specification and the route capability determines whether or not the route existed, is possible, or is not possible for various reasons. The route specification string returned from *nise.Session.find_route()* can be passed to other Switch Executive API methods (such as *nise.Session.connect()*, *nise.Session.disconnect()*, and *nise.Session.connect_and_disconnect()*) that use route specification strings.

Parameters

- **channel1** (*str*) – Channel name of one of the endpoints of the route to find. The channel name must either be a channel alias name or a name in the device/ivichannel syntax. Examples: MyChannel Switch1/R0
- **channel2** (*str*) – Channel name of one of the endpoints of the route to find. The channel name must either be a channel alias name or a name in the device/ivichannel syntax. Examples: MyChannel Switch1/R0
- **route_spec_size** (*list of int*) – The routeSpecSize is an ViInt32 that is passed by reference into the method. As an input, it is the size of the route string buffer being passed. If the route string is larger than the string buffer being passed, only the portion of the route string that can fit in the string buffer is copied into it. On return from the method, routeSpecSize holds the size required to hold the entire route string. Note that this size may be larger than the buffer size as the method always returns the size needed to hold the entire buffer. You may pass NULL for this parameter if you are not interested in the return value for routeSpecSize and routeSpec.

Return type

tuple (route_spec, path_capability)

WHERE

route_spec (str):

The fully specified route path complete with delimiting square brackets if the route exists or is possible. An example of a fully specified route string is: [A->Switch1/r0->B] Route specification strings can be directly passed to *nise.Session.connect()*, *nise.Session.disconnect()*, or *nise.Session.connect_and_disconnect()* Refer to Route Specification Strings in the NI Switch Executive Help for more information. You may pass NULL for this parameter if you are not interested in the return value. To obtain the route specification string, you should pass a buffer to this parameter. The size of the buffer required may be obtained by calling the method with NULL for this parameter and a valid ViInt32 to routeSpecSize. The routeSpecSize will contain the size needed to hold the entire route specification (including the NULL termination character). Common operation is to call the method twice. The first time you call the method you can determine the size needed to hold the route specification string. Allocate a buffer of the appropriate size and then re-call the method to obtain the entire buffer.

path_capability (*nise.PathCapability*):

The return value which expresses the capability of finding a valid route between Channel 1 and Channel 2. Refer to the table below for value descriptions. You may pass NULL for this parameter if you are not interested in the return value. Route capability might be one of the following: Path Available (1) A path between channel 1 and channel 2 is available. The route specification parameter returns a string describing the available path. Path Exists (2) A path between channel 1 and channel 2 already exists. The route specification parameter returns a string describing the existing path. Path Unsupported (3) There is no potential path between channel 1 and channel 2 given the current configuration. Resource In Use (4) There is a potential path between channel 1 and channel 2, although a resource needed to complete the path is already in use. Source Conflict (5) Channel 1 and channel 2 cannot be connected because their connection would result in an exclusion violation. Channel Not Available (6) One of the channels is not useable as an endpoint channel. Make sure that it is not marked as a reserved for routing. Channels Hardwired (7) The two channels reside on the same hardwire. An implicit path already exists.

get_all_connections

nise.Session.**get_all_connections**(*route_spec_size=[1024]*)

Returns the top-level connected routes and route groups. The route specification string returned from *nise.Session.get_all_connections()* can be passed to other Switch Executive API methods (such as *nise.Session.connect()*, *nise.Session.disconnect()*, *nise.Session.connect_and_disconnect()*, and *nise.Session.expand_route_spec()*) that use route specification strings.

Parameters

route_spec_size (*list of int*) – The routeSpecSize is an ViInt32 that is passed by reference into the method. As an input, it is the size of the route spec string buffer being passed. If the route spec string is larger than the string buffer being passed, only

the portion of the route spec string that can fit in the string buffer is copied into it. On return from the method, routeSpecSize holds the size required to hold the entire route spec string. Note that this size may be larger than the buffer size as the method always returns the size needed to hold the entire buffer. You may pass NULL for this parameter if you are not interested in the return value for routeSpecSize and routeSpec.

Return type
str

Returns
The route spec of all currently connected routes and route groups. Route specification strings can be directly passed to *nise.Session.connect()*, *nise.Session.disconnect()*, *nise.Session.connect_and_disconnect()*, or *nise.Session.expand_route_spec()* Refer to Route Specification Strings in the NI Switch Executive Help for more information. You may pass NULL for this parameter if you are not interested in the return value. To obtain the route specification string, you should pass a buffer to this parameter. The size of the buffer required may be obtained by calling the method with NULL for this parameter and a valid ViInt32 to routeSpecSize. The routeSpecSize will contain the size needed to hold the entire route specification (including the NULL termination character). Common operation is to call the method twice. The first time you call the method you can determine the size needed to hold the route specification string. Allocate a buffer of the appropriate size and then re-call the method to obtain the entire buffer.

is_connected

nise.Session.**is_connected**(*route_spec*)

Checks whether the specified routes and routes groups are connected. It returns true if connected.

Parameters
route_spec (*str*) – String describing the connections to check. The route specification strings are best summarized as a series of routes delimited by ampersands. The specified routes may be route names, route group names, or fully specified route paths delimited by square brackets. Some examples of route specification strings are: MyRoute MyRouteGroup MyRoute & MyRouteGroup [A->Switch1/r0->B] MyRoute & MyRouteGroup & [A->Switch1/r0->B] Refer to Route Specification Strings in the NI Switch Executive Help for more information.

Return type
bool

Returns
Returns TRUE if the routes and routes groups are connected or FALSE if they are not.

is_debounced

nise.Session.**is_debounced**()

Checks to see if the switching system is debounced or not. This method does not wait for debouncing to occur. It returns true if the system is fully debounced. This method is similar to the IviSwtch specific method.

Return type
bool

Returns

Returns TRUE if the system is fully debounced or FALSE if it is still settling.

wait_for_debounce

nise.Session.**wait_for_debounce**(*maximum_time_ms=hightime.timedelta(milliseconds=-1)*)

Waits for all of the switches in the NI Switch Executive virtual device to debounce. This method does not return until either the switching system is completely debounced and settled or the maximum time has elapsed and the system is not yet debounced. In the event that the maximum time elapses, the method returns an error indicating that a timeout has occurred. To ensure that all of the switches have settled, NI recommends calling *nise.Session.wait_for_debounce()* after a series of connection or disconnection operations and before taking any measurements of the signals connected to the switching system.

Parameters

maximum_time_ms (*hightime.timedelta, datetime.timedelta, or int in milliseconds*) – The amount of time to wait (in milliseconds) for the debounce to complete. A value of 0 checks for debouncing once and returns an error if the system is not debounced at that time. A value of -1 means to block for an infinite period of time until the system is debounced.

Session

Enums

Enums used in NI Switch Executive

ExpandAction

class nise.**ExpandAction**

ROUTES
Expand to routes

PATHS
Expand to paths

MulticonnectMode

class nise.**MulticonnectMode**

DEFAULT
Default

NO_MULTICONNECT
No multiconnect

MULTICONNECT
Multiconnect

OperationOrder

class nise.**OperationOrder**

BEFORE
Break before make

AFTER
Break after make

PathCapability

class nise.**PathCapability**

PATH_NEEDS_HARDWIRE
Path needs hardwire

PATH_NEEDS_CONFIG_CHANNEL
Path needs config channel

PATH_AVAILABLE
Path available

PATH_EXISTS
Path exists

PATH_UNSUPPORTED
Path Unsupported

RESOURCE_IN_USE
Resource in use

EXCLUSION_CONFLICT
Exclusion conflict

CHANNEL_NOT_AVAILABLE
Channel not available

CHANNELS_HARDWIRED
Channels hardwired

Exceptions and Warnings

Error

exception nise.errors.**Error**
Base exception type that all NI Switch Executive exceptions derive from

DriverError

exception nise.errors.**DriverError**
An error originating from the NI Switch Executive driver

UnsupportedConfigurationError

exception nise.errors.**UnsupportedConfigurationError**
An error due to using this module in an usupported platform.

DriverNotInstalledError

exception nise.errors.**DriverNotInstalledError**
An error due to using this module without the driver runtime installed.

DriverTooOldError

exception nise.errors.**DriverTooOldError**
An error due to using this module with an older version of the NI Switch Executive driver runtime.

DriverTooNewError

exception nise.errors.**DriverTooNewError**

An error due to the NI Switch Executive driver runtime being too new for this module.

InvalidRepeatedCapabilityError

exception nise.errors.**InvalidRepeatedCapabilityError**

An error due to an invalid character in a repeated capability

DriverWarning

exception nise.errors.**DriverWarning**

A warning originating from the NI Switch Executive driver

Examples

You can download all nise examples for latest version here

nise_basic_example.py

Listing 1: (nise_basic_example.py)

```
#!/usr/bin/python
import argparse
import nise
import sys


def example(virtual_device_name, connection):
    with nise.Session(virtual_device_name=virtual_device_name) as session:
        session.connect(connection)
        print(connection, ' is now connected.')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Connects the specified connection␣
↪specification', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--virtual-device', default='SwitchExecutiveExample', help=
↪'NI Switch Executive Virtual Device name')
    parser.add_argument('-c', '--connection', default='DIOToUUT', help='Connection␣
↪Specification')
    args = parser.parse_args(argsv)
    example(args.virtual_device, args.connection)


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

(continues on next page)

(continued from previous page)

```
23
24
25  def test_example():
26      example('SwitchExecutiveExample', 'DIOToUUT')
27
28
29  def test_main():
30      cmd_line = []
31      _main(cmd_line)
32
33
34  if __name__ == '__main__':
35      main()
36
37
```

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