NI-Digital Pattern Driver Python API Documentation

Release 1.4.9.dev0

NI

May 01, 2024

DOCUMENTATION

CHAPTER

ONE

ABOUT

The **nidigital** module provides a Python API for NI-Digital Pattern Driver. The code is maintained in the Open Source repository for nimi-python.

1.1 Support Policy

nidigital supports all the Operating Systems supported by NI-Digital Pattern Driver.

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 nidigital module

4.1.1 Installation

As a prerequisite to using the **nidigital** module, you must install the NI-Digital Pattern Driver runtime on your system. Visit ni.com/downloads to download the driver runtime for your devices.

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

```
$ python -m pip install nidigital
```

4.1.2 Usage

The following is a basic example of using the **nidigital** module to open a session to a digital pattern instrument, source current, and measure both voltage and current using the PPMU on selected channels.

```
import nidigital
import time

with nidigital.Session(resource_name='PXI1Slot2') as session:

    channels = 'PXI1Slot2/0,PXI1Slot2/1'

    # Configure PPMU measurements
    session.channels[channels].ppmu_aperture_time = 0.000004
    session.channels[channels].ppmu_aperture_time_units = nidigital.
↪PPMUApertureTimeUnits.SECONDS

    session.channels[channels].ppmu_output_function = nidigital.PPMUOutputFunction.
↪CURRENT

    session.channels[channels].ppmu_current_level_range = 0.000002
    session.channels[channels].ppmu_current_level = 0.000002
    session.channels[channels].ppmu_voltage_limit_high = 3.3
```

(continues on next page)

(continued from previous page)

```
    session.channels[channels].ppmu_voltage_limit_low = 0

    # Sourcing
    session.channels[channels].ppmu_source()

    # Settling time between sourcing and measuring
    time.sleep(0.01)

    # Measuring
    current_measurements = session.channels[channels].ppmu_measure(nidigital.
↪PPMUMeasurementType.CURRENT)
    voltage_measurements = session.channels[channels].ppmu_measure(nidigital.
↪PPMUMeasurementType.VOLTAGE)

    print('{:<20} {:<10} {:<10}'.format('Channel Name', 'Current', 'Voltage'))
    for channel, current, voltage in zip(channels.split(','), current_measurements,␣
↪voltage_measurements):
        print('{:<20} {:<10f} {:<10f}'.format(channel, current, voltage))

    # Disconnect all channels using programmable onboard switching
    session.channels[channels].selected_function = nidigital.SelectedFunction.DISCONNECT
```

Other usage examples can be found on GitHub.

4.1.3 API Reference

Session

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

Creates and returns a new session to the specified digital pattern instrument to use in all subsequent method calls. To place the instrument in a known startup state when creating a new session, set the reset parameter to True, which is equivalent to calling the *nidigital.Session.reset()* method immediately after initializing the session.

Parameters

- **resource_name** (*str*) – The specified resource name shown in Measurement & Automation Explorer (MAX) for a digital pattern instrument, for example, PXI1Slot3, where PXI1Slot3 is an instrument resource name. **resourceName** can also be a logical IVI name. This parameter accepts a comma-delimited list of strings in the form PXI1Slot2,PXI1Slot3, where `PXI1Slot2` is one instrument resource name and `PXI1Slot3` is another. When including more than one digital pattern instrument in the comma-delimited list of strings, list the instruments in the same order they appear in the pin map.

  Note You only can specify multiple instruments of the same model. For example, you can list two PXIe-6570s but not a PXIe-6570 and PXIe-6571. The instruments must be in the same chassis.

- **id_query** (*bool*) – A Boolean that verifies that the digital pattern instrument you initialize is supported by NI-Digital. NI-Digital automatically performs this query, so setting this parameter is not necessary.

- **reset_device** (*bool*) – A Boolean that specifies whether to reset a digital pattern instrument to a known state when the session is initialized. Setting the **resetDevice** value to True is equivalent to calling the *nidigital.Session.reset()* method immediately after initializing the session.
- **options** (*dict*) – Specifies the initial value of certain properties for the session. The syntax for **options** is a dictionary of properties with an assigned value. For example:

  { 'simulate': False }

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

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

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

- **grpc_options** (nidigital.GrpcSessionOptions) – MeasurementLink gRPC session options

Methods

abort

nidigital.Session.**abort**()

Stops bursting the pattern.

abort_keep_alive

nidigital.Session.**abort_keep_alive**()

Stops the keep alive pattern if it is currently running. If a pattern burst is in progress, the method aborts the pattern burst. If you start a new pattern burst while a keep alive pattern is running, the keep alive pattern runs to the last keep alive vector, and the new pattern burst starts on the next cycle.

apply_levels_and_timing

nidigital.Session.**apply_levels_and_timing**(*levels_sheet*, *timing_sheet*, *initial_state_high_pins=None*, *initial_state_low_pins=None*, *initial_state_tristate_pins=None*)

Applies digital levels and timing values defined in previously loaded levels and timing sheets. When applying a levels sheet, only the levels specified in the sheet are affected. Any levels not specified in the sheet remain unchanged. When applying a timing sheet, all existing time sets are deleted before the new time sets are loaded.

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

Example: `my_session.sites[ ... ].apply_levels_and_timing()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.apply_levels_and_timing()`

Parameters

- **levels_sheet** (*str*) – Name of the levels sheet to apply. Use the name of the sheet or pass the absolute file path you use in the *nidigital.Session.load_specifications_levels_and_timing()* method. The name of the levels sheet is the file name without the directory and file extension.
- **timing_sheet** (*str*) – Name of the timing sheet to apply. Use the name of the sheet or pass the absolute file path that you use in the *nidigital.Session.load_specifications_levels_and_timing()* method. The name of the timing sheet is the file name without the directory and file extension.
- **initial_state_high_pins** (*basic sequence types or str*) – Comma-delimited list of pins, pin groups, or channels to initialize to a high state.
- **initial_state_low_pins** (*basic sequence types or str*) – Comma-delimited list of pins, pin groups, or channels to initialize to a low state.
- **initial_state_tristate_pins** (*basic sequence types or str*) – Comma-delimited list of pins, pin groups, or channels to initialize to a non-drive state (X)

apply_tdr_offsets

nidigital.Session.**apply_tdr_offsets**(*offsets*)

Applies the correction for propagation delay offsets to a digital pattern instrument. Use this method to apply TDR offsets that are stored from a past measurement or are measured by means other than the *nidigital.Session.tdr()* method. Also use this method to apply correction for offsets if the **applyOffsets** input of the *nidigital.Session.tdr()* method was set to False at the time of measurement.

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

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

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

Example: `my_session.apply_tdr_offsets()`

Parameters

offsets (*basic sequence of hightime.timedelta, datetime.timedelta, or float in seconds*) – TDR offsets to apply, in seconds. Specify

an offset for each pin or channel in the repeated capabilities. If the repeated capabilities contain pin names, you must specify offsets for each site in the channel map per pin.

burst_pattern

nidigital.Session.**burst_pattern**(*start_label*, *select_digital_function=True*, *wait_until_done=True*, *timeout=hightime.timedelta(seconds=10.0)*)

Uses the start_label you specify to burst the pattern on the sites you specify. If you specify wait_until_done as True, waits for the burst to complete, and returns comparison results for each site.

Digital pins retain their state at the end of a pattern burst until the first vector of the pattern burst, a call to *nidigital.Session.write_static()*, or a call to *nidigital.Session.apply_levels_and_timing()*.

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

Example: `my_session.sites[ ... ].burst_pattern()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.burst_pattern()`

Parameters

- **start_label** (*str*) – Pattern name or exported pattern label from which to start bursting the pattern.
- **select_digital_function** (*bool*) – A Boolean that specifies whether to select the digital method for the pins in the pattern prior to bursting.
- **wait_until_done** (*bool*) – A Boolean that indicates whether to wait until the bursting is complete.
- **timeout** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Maximum time (in seconds) allowed for this method to complete. If this method does not complete within this time interval, this method returns an error.

Return type

{ int: bool, int: bool, ... }

Returns

Dictionary where each key is a site number and value is pass/fail, if wait_until_done is specified as True. Else, None.

clock_generator_abort

nidigital.Session.**clock_generator_abort**()

Stops clock generation on the specified channel(s) or pin(s) and pin group(s).

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

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

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

Example: `my_session.clock_generator_abort()`

---

clock_generator_generate_clock

nidigital.Session.**clock_generator_generate_clock**(*frequency*, *select_digital_function=True*)

Configures clock generator frequency and initiates clock generation on the specified channel(s) or pin(s) and pin group(s).

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

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

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

Example: `my_session.clock_generator_generate_clock()`

Parameters

- **frequency** (*float*) – The frequency of the clock generation, in Hz.
- **select_digital_function** (*bool*) – A Boolean that specifies whether to select the digital method for the pins specified prior to starting clock generation.

close

nidigital.Session.**close**()

Closes the specified instrument session to a digital pattern instrument, aborts pattern execution, and unloads pattern memory. The channels on a digital pattern instrument remain in their current state.

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

---

commit

nidigital.Session.**commit**()

Applies all previously configured pin levels, termination modes, clocks, triggers, and pattern timing to a digital pattern instrument. If you do not call the *nidigital.Session.commit()* method, then the initiate method or the *nidigital.Session.burst_pattern()* method will implicitly call this method for you. Calling this method moves the session from the Uncommitted state to the Committed state.

configure_active_load_levels

nidigital.Session.**configure_active_load_levels**(*iol*, *ioh*, *vcom*)

Configures I_{OL}, I_{OH}, and V_{COM} levels for the active load on the pins you specify. The DUT sources or sinks current based on the level values. To enable active load, set the termination mode to *ACTIVE_LOAD*. To disable active load, set the termination mode of the instrument to *HIGH_Z* or *VTERM*.

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

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

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

Example: `my_session.configure_active_load_levels()`

Parameters

- **iol** (*float*) – Maximum current that the DUT sinks while outputting a voltage below V_{COM}.
- **ioh** (*float*) – Maximum current that the DUT sources while outputting a voltage above V_{COM}.
- **vcom** (*float*) – Commutating voltage level at which the active load circuit switches between sourcing current and sinking current.

configure_pattern_burst_sites

nidigital.Session.**configure_pattern_burst_sites**()

Configures which sites burst the pattern on the next call to the initiate method. The pattern burst sites can also be modified through the repeated capabilities for the *nidigital.Session.burst_pattern()* method. If a site has been disabled through the *nidigital.Session.disable_sites()* method, the site does not burst a pattern even if included in the pattern burst sites.

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

Example: `my_session.sites[ ... ].configure_pattern_burst_sites()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_pattern_burst_sites()`

---

configure_time_set_compare_edges_strobe

`nidigital.Session.`**`configure_time_set_compare_edges_strobe`**(*time_set_name*, *strobe_edge*)

Configures the strobe edge time for the specified pins. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_compare_edges_strobe()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_compare_edges_strobe()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **strobe_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Time when the comparison happens within a vector period.

configure_time_set_compare_edges_strobe2x

`nidigital.Session.`**`configure_time_set_compare_edges_strobe2x`**(*time_set_name*, *strobe_edge*, *strobe2_edge*)

Configures the compare strobes for the specified pins in the time set, including the 2x strobe. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_compare_edges_strobe2x()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_compare_edges_strobe2x()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **strobe_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Time when the comparison happens within a vector period.
- **strobe2_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Time when the comparison happens for the second DUT cycle within a vector period.

configure_time_set_drive_edges

nidigital.Session.**configure_time_set_drive_edges**(*time_set_name*, *format*, *drive_on_edge*, *drive_data_edge*, *drive_return_edge*, *drive_off_edge*)

Configures the drive format and drive edge placement for the specified pins. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_drive_edges()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_drive_edges()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **format** (*nidigital.DriveFormat*) – Drive format of the time set.
  - *NR*: Non-return.
  - *RL*: Return to low.
  - *RH*: Return to high.
  - *SBC*: Surround by complement.
- **drive_on_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period for turning on the pin driver.This option applies only when the prior vector left the pin in a non-drive pin state (L, H, X, V, M, E). For the SBC format, this option specifies the delay from the beginning of the vector period at which the complement of the pattern value is driven.
- **drive_data_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pattern data is driven to the pattern value.The ending state from the previous vector persists until this point.

- **drive_return_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pin changes from the pattern data to the return value, as specified in the format.
- **drive_off_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period to turn off the pin driver when the next vector period uses a non-drive symbol (L, H, X, V, M, E).

configure_time_set_drive_edges2x

nidigital.Session.**configure_time_set_drive_edges2x**(*time_set_name*, *format*, *drive_on_edge*, *drive_data_edge*, *drive_return_edge*, *drive_off_edge*, *drive_data2_edge*, *drive_return2_edge*)

Configures the drive edges of the pins in the time set, including 2x edges. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_drive_edges2x()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_drive_edges2x()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **format** (*nidigital.DriveFormat*) – Drive format of the time set.
  - *NR*: Non-return.
  - *RL*: Return to low.
  - *RH*: Return to high.
  - *SBC*: Surround by complement.
- **drive_on_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period for turning on the pin driver.This option applies only when the prior vector left the pin in a non-drive pin state (L, H, X, V, M, E). For the SBC format, this option specifies the delay from the beginning of the vector period at which the complement of the pattern value is driven.
- **drive_data_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period

until the pattern data is driven to the pattern value.The ending state from the previous vector persists until this point.

- **drive_return_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pin changes from the pattern data to the return value, as specified in the format.
- **drive_off_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period to turn off the pin driver when the next vector period uses a non-drive symbol (L, H, X, V, M, E).
- **drive_data2_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pattern data in the second DUT cycle is driven to the pattern value.
- **drive_return2_edge** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Delay, in seconds, from the beginning of the vector period until the pin changes from the pattern data in the second DUT cycle to the return value, as specified in the format.

configure_time_set_drive_format

nidigital.Session.**configure_time_set_drive_format**(*time_set_name*, *drive_format*)

Configures the drive format for the pins specified in the **pinList**. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_drive_format()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_drive_format()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **drive_format** (*nidigital.DriveFormat*) – Drive format of the time set.
  - *NR*: Non-return.
  - *RL*: Return to low.
  - *RH*: Return to high.
  - *SBC*: Surround by complement.

configure_time_set_edge

nidigital.Session.**configure_time_set_edge**(*time_set_name*, *edge*, *time*)

Configures the edge placement for the pins specified in the pin list. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_edge()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_edge()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **edge** (*nidigital.TimeSetEdgeType*) – Name of the edge.
  - *DRIVE_ON*
  - *DRIVE_DATA*
  - *DRIVE_RETURN*
  - *DRIVE_OFF*
  - *COMPARE_STROBE*
  - *DRIVE_DATA2*
  - *DRIVE_RETURN2*
  - *COMPARE_STROBE2*
- **time** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – The time from the beginning of the vector period in which to place the edge.

configure_time_set_edge_multiplier

nidigital.Session.**configure_time_set_edge_multiplier**(*time_set_name*, *edge_multiplier*)

Configures the edge multiplier of the pins in the time set. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

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

Example: `my_session.pins[ ... ].configure_time_set_edge_multiplier()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.configure_time_set_edge_multiplier()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **edge_multiplier** (*int*) – The specified edge multiplier for the pins in the pin list.

configure_time_set_period

nidigital.Session.**configure_time_set_period**(*time_set_name*, *period*)

Configures the period of a time set. Use this method to modify time set values after applying a timing sheet with the *nidigital.Session.apply_levels_and_timing()* method, or to create time sets programmatically without the use of timing sheets. This method does not modify the timing sheet file or the timing sheet contents that will be used in future calls to *nidigital.Session.apply_levels_and_timing()*; it only affects the values of the current timing context.

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **period** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Period for this time set, in seconds.

configure_voltage_levels

nidigital.Session.**configure_voltage_levels**(*vil*, *vih*, *vol*, *voh*, *vterm*)

Configures voltage levels for the pins you specify.

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

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

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

Example: `my_session.configure_voltage_levels()`

Parameters

- **vil** (*float*) – Voltage that the instrument will apply to the input of the DUT when the pin driver drives a logic low (0).
- **vih** (*float*) – Voltage that the instrument will apply to the input of the DUT when the test instrument drives a logic high (1).

- **vol** (*float*) – Output voltage below which the comparator on the pin driver interprets a logic low (L).
- **voh** (*float*) – Output voltage above which the comparator on the pin driver interprets a logic high (H).
- **vterm** (*float*) – Termination voltage the instrument applies during non-drive cycles when the termination mode is set to V_{term}. The instrument applies the termination voltage through a 50 ohm parallel termination resistance.

create_capture_waveform_from_file_digicapture

nidigital.Session.**create_capture_waveform_from_file_digicapture**(*waveform_name*, *waveform_file_path*)

Creates a capture waveform with the configuration information from a Digicapture file generated by the Digital Pattern Editor.

Parameters

- **waveform_name** (*str*) – Waveform name you want to use. You must specify waveform_name if the file contains multiple waveforms. Use the waveform_name with the capture_start opcode in your pattern.
- **waveform_file_path** (*str*) – Absolute file path to the capture waveform file (.digicapture) you want to load.

create_capture_waveform_parallel

nidigital.Session.**create_capture_waveform_parallel**(*waveform_name*)

Sets the capture waveform settings for parallel acquisition. Settings apply across all sites if multiple sites are configured in the pin map. You cannot reconfigure settings after waveforms are created.

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

Example: `my_session.pins[ ... ].create_capture_waveform_parallel()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.create_capture_waveform_parallel()`

Parameters

waveform_name (*str*) – Waveform name you want to use. Use the waveform_name with the capture_start opcode in your pattern.

create_capture_waveform_serial

nidigital.Session.**create_capture_waveform_serial**(*waveform_name*, *sample_width*, *bit_order*)

Sets the capture waveform settings for serial acquisition. Settings apply across all sites if multiple sites are configured in the pin map. You cannot reconfigure settings after waveforms are created.

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

Example: `my_session.pins[ ... ].create_capture_waveform_serial()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.create_capture_waveform_serial()`

Parameters

- **waveform_name** (*str*) – Waveform name you want to use. Use the waveform_name with the capture_start opcode in your pattern.
- **sample_width** (*int*) – Width in bits of each serial sample. Valid values are between 1 and 32.
- **bit_order** (*nidigital.BitOrder*) – Order in which to shift the bits.
  - *MSB*: Specifies the bit order by most significant bit first.
  - *LSB*: Specifies the bit order by least significant bit first.

create_source_waveform_from_file_tdms

nidigital.Session.**create_source_waveform_from_file_tdms**(*waveform_name*, *waveform_file_path*, *write_waveform_data=True*)

Creates a source waveform with configuration information from a TDMS file generated by the Digital Pattern Editor. It also optionally writes waveform data from the file.

Parameters

- **waveform_name** (*str*) – The waveform name you want to use from the file. You must specify waveform_name if the file contains multiple waveforms. Use the waveform_name with the source_start opcode in your pattern.
- **waveform_file_path** (*str*) – Absolute file path to the load source waveform file (.tdms).
- **write_waveform_data** (*bool*) – A Boolean that writes waveform data to source memory if True and the waveform data is in the file.

create_source_waveform_parallel

nidigital.Session.**create_source_waveform_parallel**(*waveform_name*, *data_mapping*)

Sets the source waveform settings required for parallel sourcing. Settings apply across all sites if multiple sites are configured in the pin map. You cannot reconfigure settings after waveforms are created.

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

Example: `my_session.pins[ ... ].create_source_waveform_parallel()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.create_source_waveform_parallel()`

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **data_mapping** (*nidigital.SourceDataMapping*) – Parameter that specifies how to map data on multiple sites.
  - *BROADCAST*: Broadcasts the waveform you specify to all sites.
  - *SITE_UNIQUE*: Sources unique waveform data to each site.

create_source_waveform_serial

nidigital.Session.**create_source_waveform_serial**(*waveform_name*, *data_mapping*, *sample_width*, *bit_order*)

Sets the source waveform settings required for serial sourcing. Settings apply across all sites if multiple sites are configured in the pin map. You cannot reconfigure settings after waveforms are created.

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

Example: `my_session.pins[ ... ].create_source_waveform_serial()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.create_source_waveform_serial()`

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **data_mapping** (*nidigital.SourceDataMapping*) – Parameter that specifies how to map data on multiple sites.
  - *BROADCAST*: Broadcasts the waveform you specify to all sites.

- *SITE_UNIQUE*: Sources unique waveform data to each site.

- **sample_width** (*int*) – Width in bits of each serial sample. Valid values are between 1 and 32.

- **bit_order** (*nidigital.BitOrder*) – Order in which to shift the bits.
  - *MSB*: Specifies the bit order by most significant bit first.
  - *LSB*: Specifies the bit order by least significant bit first.

create_time_set

nidigital.Session.**create_time_set**(*name*)

Creates a time set with the name that you specify. Use this method when you want to create time sets programmatically rather than with a timing sheet.

Parameters

name (*str*) – The specified name of the new time set.

delete_all_time_sets

nidigital.Session.**delete_all_time_sets**()

Deletes all time sets from instrument memory.

disable_sites

nidigital.Session.**disable_sites**()

Disables specified sites. Disabled sites are not included in pattern bursts initiated by the initiate method or the *nidigital.Session.burst_pattern()* method, even if the site is specified in the list of pattern burst sites in *nidigital.Session.configure_pattern_burst_sites()* method or in the repeated capabilities for the *nidigital.Session.burst_pattern()* method. Additionally, if you specify a list of pin or pin group names in repeated capabilities in any NI-Digital method, digital pattern instrument channels mapped to disabled sites are not affected by the method. The methods that return per-pin data, such as the *nidigital.Session.ppmu_measure()* method, do not return data for channels mapped to disabled sites. The digital pattern instrument channels mapped to the sites specified are left in their current state. NI TestStand Semiconductor Module requires all sites to always be enabled, and manages the set of active sites without disabling the sites in the digital instrument session. Do not use this method with the Semiconductor Module.

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

Example: `my_session.sites[ ... ].disable_sites()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.disable_sites()`

---

enable_sites

nidigital.Session.**enable_sites**()

Enables the sites you specify. All sites are enabled by default.

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

Example: `my_session.sites[ ... ].enable_sites()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.enable_sites()`

---

fetch_capture_waveform

nidigital.Session.**fetch_capture_waveform**(*waveform_name*, *samples_to_read*, *timeout=hightime.timedelta(seconds=10.0)*)

Returns dictionary where each key is a site number and value is a collection of digital states representing capture waveform data

---

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

Example: `my_session.sites[ ... ].fetch_capture_waveform()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.fetch_capture_waveform()`

Parameters

- **waveform_name** (*str*) – Waveform name you create with the create capture waveform method. Use the waveform_name parameter with capture_start opcode in your pattern.
- **samples_to_read** (*int*) – Number of samples to fetch.
- **timeout** (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Maximum time (in seconds) allowed for this method to complete. If this method does not complete within this time interval, this method returns an error.

Return type

{ int: memoryview of array.array of unsigned int, int: memoryview of array.array of unsigned int, … }

Returns

Dictionary where each key is a site number and value is a collection of digital states representing capture waveform data

fetch_history_ram_cycle_information

nidigital.Session.**fetch_history_ram_cycle_information**(*position*, *samples_to_read*)

Returns the pattern information acquired for the specified cycles.

If the pattern is using the edge multiplier feature, cycle numbers represent tester cycles, each of which may consist of multiple DUT cycles. When using pins with mixed edge multipliers, pins may return *PIN_STATE_NOT_ACQUIRED* for DUT cycles where those pins do not have edges defined.

Site number on which to retrieve pattern information must be specified via sites repeated capability. The method returns an error if more than one site is specified.

Pins for which to retrieve pattern information must be specified via pins repeated capability. If pins are not specified, pin list from the pattern containing the start label is used. Call *nidigital.Session.get_pattern_pin_names()* with the start label to retrieve the pins associated with the pattern burst:

```
session.sites[0].pins['PinA', 'PinB'].fetch_history_ram_cycle_
↪information(0, -1)
```

Note: Before bursting a pattern, you must configure the History RAM trigger and specify which cycles to acquire.

nidigital.Session.history_ram_trigger_type should be used to specify the trigger condition on which History RAM starts acquiring pattern information.

If History RAM trigger is configured as *CYCLE_NUMBER*, *nidigital.Session.cycle_number_history_ram_trigger_cycle_number* should be used to specify the cycle number on which History RAM starts acquiring pattern information.

If History RAM trigger is configured as *PATTERN_LABEL*, *nidigital.Session.pattern_label_history_ram_trigger_label* should be used to specify the pattern label from which to start acquiring pattern information. *nidigital.Session.pattern_label_history_ram_trigger_vector_offset* should be used to specify the number of vectors following the specified pattern label from which to start acquiring pattern information. *nidigital.Session.pattern_label_history_ram_trigger_cycle_offset* should be used to specify the number of cycles following the specified pattern label and vector offset from which to start acquiring pattern information.

For all History RAM trigger conditions, *nidigital.Session.history_ram_pretrigger_samples* should be used to specify the number of samples to acquire before the trigger conditions are met. If you configure History RAM to only acquire failed cycles, you must set *nidigital.Session.history_ram_pretrigger_samples* to 0.

nidigital.Session.history_ram_cycles_to_acquire should be used to specify which cycles History RAM acquires after the trigger conditions are met.

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

Example: `my_session.pins[ ... ].fetch_history_ram_cycle_information()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.fetch_history_ram_cycle_information()`

Parameters

- **position** (*int*) – Sample index from which to start fetching pattern information.
- **samples_to_read** (*int*) – Number of samples to fetch. A value of -1 specifies to fetch all available samples.

Return type
list of HistoryRAMCycleInformation

Returns

Returns a list of class instances with the following information about each pattern cycle:

- **pattern_name** (str) Name of the pattern for the acquired cycle.
- **time_set_name** (str) Time set for the acquired cycle.
- **vector_number** (int) Vector number within the pattern for the acquired cycle. Vector numbers start at 0 from the beginning of the pattern.
- **cycle_number** (int) Cycle number acquired by this History RAM sample. Cycle numbers start at 0 from the beginning of the pattern burst.
- **scan_cycle_number** (int) Scan cycle number acquired by this History RAM sample. Scan cycle numbers start at 0 from the first cycle of the scan vector. Scan cycle numbers are -1 for cycles that do not have a scan opcode.
- **expected_pin_states** (list of list of enums.PinState) Pin states as expected by the loaded pattern in the order specified in the pin list. Pins without defined edges in the specified DUT cycle will have a value of *PIN_STATE_NOT_ACQUIRED*. Length of the outer list will be equal to the value of edge multiplier for the given vector. Length of the inner list will be equal to the number of pins requested.
- **actual_pin_states** (list of list of enums.PinState) Pin states acquired by History RAM in the order specified in the pin list. Pins without defined edges in the specified DUT cycle will have a value of *PIN_STATE_NOT_ACQUIRED*. Length of the outer list will be equal to the value of edge multiplier for the given vector. Length of the inner list will be equal to the number of pins requested.
- **per_pin_pass_fail** (list of list of bool) Pass fail information for pins in the order specified in the pin list. Pins without defined edges in the specified DUT cycle will have a value of pass (True). Length of the outer list will be equal to the value of edge multiplier for the given vector. Length of the inner list will be equal to the number of pins requested.

frequency_counter_measure_frequency

nidigital.Session.**frequency_counter_measure_frequency**()

Measures the frequency on the specified channel(s) over the specified measurement time. All channels in the repeated capabilities should have the same measurement time.

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

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

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

Example: `my_session.frequency_counter_measure_frequency()`

Return type
list of float

Returns
The returned frequency counter measurement, in Hz.This method returns -1 if the measurement is invalid for the channel.

get_channel_names

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

Returns a list of channel names for given channel indices.

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

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

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

Return type
list of str

Returns
The channel name(s) at the specified indices.

get_fail_count

`nidigital.Session.`**`get_fail_count`**`()`

Returns the comparison fail count for pins in the repeated capabilities.

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

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

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

Example: `my_session.get_fail_count()`

Return type
list of int

Returns

Number of failures in an array. If a site is disabled or not enabled for burst, the method does not return data for that site. You can also use the *nidigital.Session.get_pin_results_pin_information()* method to obtain a sorted list of returned sites and channels.

get_history_ram_sample_count

nidigital.Session.**get_history_ram_sample_count**()

Returns the number of samples History RAM acquired on the last pattern burst.

Note: Before bursting a pattern, you must configure the History RAM trigger and specify which cycles to acquire.

nidigital.Session.history_ram_trigger_type should be used to specify the trigger condition on which History RAM starts acquiring pattern information.

If History RAM trigger is configured as *CYCLE_NUMBER*, *nidigital.Session.cycle_number_history_ram_trigger_cycle_number* should be used to specify the cycle number on which History RAM starts acquiring pattern information.

If History RAM trigger is configured as *PATTERN_LABEL*, *nidigital.Session.pattern_label_history_ram_trigger_label* should be used to specify the pattern label from which to start acquiring pattern information. *nidigital.Session.pattern_label_history_ram_trigger_vector_offset* should be used to specify the number of vectors following the specified pattern label from which to start acquiring pattern information. *nidigital.Session.pattern_label_history_ram_trigger_cycle_offset* should be used to specify the number of cycles following the specified pattern label and vector offset from which to start acquiring pattern information.

For all History RAM trigger conditions, *nidigital.Session.history_ram_pretrigger_samples* should be used to specify the number of samples to acquire before the trigger conditions are met. If you configure History RAM to only acquire failed cycles, you must set *nidigital.Session.history_ram_pretrigger_samples* to 0.

nidigital.Session.history_ram_cycles_to_acquire should be used to specify which cycles History RAM acquires after the trigger conditions are met.

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

Example: `my_session.sites[ ... ].get_history_ram_sample_count()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.get_history_ram_sample_count()`

Return type

int

Returns

The returned number of samples that History RAM acquired.

get_pattern_pin_names

nidigital.Session.**get_pattern_pin_names**(*start_label*)

Returns the pattern pin list.

Parameters

start_label (*str*) – Pattern name or exported pattern label from which to get the pin names that the pattern references.

Return type

list of str

Returns

List of pins referenced by the pattern with the **startLabel**.

get_pin_results_pin_information

nidigital.Session.**get_pin_results_pin_information**()

Returns the pin names, site numbers, and channel names that correspond to per-pin data read from the digital pattern instrument. The method returns pin information in the same order as values read using the *nidigital.Session.read_static()* method, *nidigital.Session.ppmu_measure()* method, and *nidigital.Session.get_fail_count()* method. Use this method to match values the previously listed methods return with pins, sites, and instrument channels.

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

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

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

Example: `my_session.get_pin_results_pin_information()`

Return type

list of PinInfo

Returns

List of named tuples with fields:

- **pin_name** (str)
- **site_number** (int)
- **channel_name** (str)

get_site_pass_fail

nidigital.Session.**get_site_pass_fail**()

Returns dictionary where each key is a site number and value is pass/fail

---

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

Example: `my_session.sites[ ... ].get_site_pass_fail()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.get_site_pass_fail()`

Return type
{ int: bool, int: bool, ... }

Returns
Dictionary where each key is a site number and value is pass/fail

get_time_set_drive_format

nidigital.Session.**get_time_set_drive_format**(*time_set_name*)

Returns the drive format of a pin in the specified time set.

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

Example: `my_session.pins[ ... ].get_time_set_drive_format()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.get_time_set_drive_format()`

Parameters
time_set_name (*str*) – The specified time set name.

Return type
nidigital.DriveFormat

Returns
Returned drive format of the time set for the specified pin.

get_time_set_edge

nidigital.Session.**get_time_set_edge**(*time_set_name*, *edge*)

Returns the edge time of a pin in the specified time set.

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

Example: `my_session.pins[ ... ].get_time_set_edge()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.get_time_set_edge()`

Parameters

- **time_set_name** (*str*) – The specified time set name.
- **edge** (*nidigital.TimeSetEdgeType*) – Name of the edge.
  - *DRIVE_ON*
  - *DRIVE_DATA*
  - *DRIVE_RETURN*
  - *DRIVE_OFF*
  - *COMPARE_STROBE*
  - *DRIVE_DATA2*
  - *DRIVE_RETURN2*
  - *COMPARE_STROBE2*

Return type
hightime.timedelta

Returns
Time from the beginning of the vector period in which to place the edge.

get_time_set_edge_multiplier

nidigital.Session.**get_time_set_edge_multiplier**(*time_set_name*)

Returns the edge multiplier of the specified time set.

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

Example: `my_session.pins[ ... ].get_time_set_edge_multiplier()`

To call the method on all pins, you can call it directly on the *nidigital.Session*.

Example: `my_session.get_time_set_edge_multiplier()`

Parameters
time_set_name (*str*) – The specified time set name.

Return type
int

Returns
Returned edge multiplier of the time set for the specified pin.

get_time_set_period

nidigital.Session.**get_time_set_period**(*time_set_name*)

Returns the period of the specified time set.

Parameters
time_set_name (*str*) – The specified time set name.

Return type
hightime.timedelta

Returns
Returned period, in seconds, that the edge is configured to.

initiate

nidigital.Session.**initiate**()

Starts bursting the pattern configured by *nidigital.Session.start_label*, causing the NI-Digital session to be committed. To stop the pattern burst, call *nidigital.Session.abort()*. If keep alive pattern is bursting when *nidigital.Session.abort()* is called or upon exiting the context manager, keep alive pattern will not be stopped. To stop the keep alive pattern, call *nidigital.Session.abort_keep_alive()*.

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

---

is_done

nidigital.Session.**is_done**()

Checks the hardware to determine if the pattern burst has completed or if any errors have occurred.

Return type
bool

Returns
A Boolean that indicates whether the pattern burst completed.

is_site_enabled

nidigital.Session.**is_site_enabled**()

Checks if a specified site is enabled.

Note: The method returns an error if more than one site is specified.

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

Example: `my_session.sites[ ... ].is_site_enabled()`

To call the method on all sites, you can call it directly on the *nidigital.Session*.

Example: `my_session.is_site_enabled()`

Return type
bool

Returns
Boolean value that returns whether the site is enabled or disabled.

load_pattern

nidigital.Session.**load_pattern**(*file_path*)

Loads the specified pattern file.

Parameters
file_path (*str*) – Absolute file path of the binary .digipat pattern file to load. Specify the pattern to burst using *nidigital.Session.start_label* or the start_label parameter of the *nidigital.Session.burst_pattern()* method.

load_pin_map

nidigital.Session.**load_pin_map**(*file_path*)

Loads a pin map file. You can load only a single pin and channel map file during an NI-Digital Pattern Driver session. To switch pin maps, create a new session or call the *nidigital.Session.reset()* method.

Parameters
file_path (*str*) – Absolute file path to a pin map file created with the Digital Pattern Editor or the NI TestStand Semiconductor Module.

load_specifications_levels_and_timing

nidigital.Session.**load_specifications_levels_and_timing**(*specifications_file_paths=None*, *levels_file_paths=None*, *timing_file_paths=None*)

Loads settings in specifications, levels, and timing sheets. These settings are not applied to the digital pattern instrument until *nidigital.Session.apply_levels_and_timing()* is called.

If the levels and timing sheets contains formulas, they are evaluated at load time. If the formulas refer to variables, the specifications sheets that define those variables must be loaded either first, or at the same time as the levels and timing sheets.

Parameters

- **specifications_file_paths** (*str or basic sequence of str*) – Absolute file path of one or more specifications files.
- **levels_file_paths** (*str or basic sequence of str*) – Absolute file path of one or more levels sheet files.
- **timing_file_paths** (*str or basic sequence of str*) – Absolute file path of one or more timing sheet files.

lock

nidigital.Session.**lock**()

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

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

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

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

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

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

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

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

Return type
context manager

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

ppmu_measure

nidigital.Session.**ppmu_measure**(*measurement_type*)

Instructs the PPMU to measure voltage or current. This method can be called to take a voltage measurement even if the pin method is not set to PPMU.

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

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

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

Example: `my_session.ppmu_measure()`

Parameters
measurement_type (*nidigital.PPMUMeasurementType*) – Parameter that specifies whether the PPMU measures voltage or current from the DUT.

- *CURRENT*: The PPMU measures current from the DUT.
- *VOLTAGE*: The PPMU measures voltage from the DUT.

Return type
list of float

Returns
The returned array of measurements in the order you specify in the repeated capabilities. If a site is disabled, the method does not return data for that site. You can also use the *nidigital.Session.get_pin_results_pin_information()* method to obtain a sorted list of returned sites and channels.

ppmu_source

nidigital.Session.**ppmu_source**()

Starts sourcing voltage or current from the PPMU. This method automatically selects the PPMU method. Changes to PPMU source settings do not take effect until you call this method. If you modify source settings after you call this method, you must call this method again for changes in the configuration to take effect.

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

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

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

Example: `my_session.ppmu_source()`

---

read_sequencer_flag

`nidigital.Session.`**`read_sequencer_flag`**(*flag*)

Reads the state of a pattern sequencer flag. Use pattern sequencer flags to coordinate execution between the pattern sequencer and a runtime test program.

Parameters

flag (*nidigital.SequencerFlag*) – The pattern sequencer flag you want to read.

- *FLAG0* ("seqflag0"): Reads pattern sequencer flag 0.
- *FLAG1* ("seqflag1"): Reads pattern sequencer flag 1.
- *FLAG2* ("seqflag2"): Reads pattern sequencer flag 2.
- *FLAG3* ("seqflag3"): Reads pattern sequencer flag 3.

Return type

bool

Returns

A Boolean that indicates the state of the pattern sequencer flag you specify.

read_sequencer_register

`nidigital.Session.`**`read_sequencer_register`**(*reg*)

Reads the value of a pattern sequencer register. Use pattern sequencer registers to pass numeric values between the pattern sequencer and a runtime test program. For example, you can use this method to read a register modified by the write_reg opcode during a pattern burst.

Parameters

reg (*nidigital.SequencerRegister*) – The sequencer register to read from.

- *REGISTER0* ("reg0"): Reads sequencer register 0.
- *REGISTER1* ("reg1"): Reads sequencer register 1.
- *REGISTER2* ("reg2"): Reads sequencer register 2.
- *REGISTER3* ("reg3"): Reads sequencer register 3.
- *REGISTER4* ("reg4"): Reads sequencer register 4.
- *REGISTER5* ("reg5"): Reads sequencer register 5.
- *REGISTER6* ("reg6"): Reads sequencer register 6.
- *REGISTER7* ("reg7"): Reads sequencer register 7.
- *REGISTER8* ("reg8"): Reads sequencer register 8.
- *REGISTER9* ("reg9"): Reads sequencer register 9.
- *REGISTER10* ("reg10"): Reads sequencer register 10.
- *REGISTER11* ("reg11"): Reads sequencer register 11.

- *REGISTER12* ("reg12"): Reads sequencer register 12.
- *REGISTER13* ("reg13"): Reads sequencer register 13.
- *REGISTER14* ("reg14"): Reads sequencer register 14.
- *REGISTER15* ("reg15"): Reads sequencer register 15.

Return type
int

Returns
Value read from the sequencer register.

read_static

nidigital.Session.**read_static**()

Reads the current state of comparators for pins you specify in the repeated capabilities. If there are uncommitted changes to levels or the termination mode, this method commits the changes to the pins.

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

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

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

Example: `my_session.read_static()`

Return type
list of *nidigital.PinState*

Returns

The returned array of pin states read from the channels in the repeated capabilities. Data is returned in the order you specify in the repeated capabilities. If a site is disabled, the method does not return data for that site. You can also use the *nidigital.Session.get_pin_results_pin_information()* method to obtain a sorted list of returned sites and channels.

- *L*: The comparators read a logic low pin state.
- *H*: The comparators read a logic high pin state.
- *M*: The comparators read a midband pin state.
- *V*: The comparators read a value that is above VOH and below VOL, which can occur when you set VOL higher than VOH.

reset

nidigital.Session.**reset**()

Returns a digital pattern instrument to a known state. This method performs the following actions:

- Aborts pattern execution.
- Clears pin maps, time sets, source and capture waveforms, and patterns.
- Resets all properties to default values, including the *nidigital.Session.selected_function* property that is set to *DISCONNECT*, causing the I/O switches to open.
- Stops exporting all external signals and events.

reset_device

nidigital.Session.**reset_device**()

Returns a digital pattern instrument to a known state. This method performs the following actions:

- Aborts pattern execution.
- Clears pin maps, time sets, source and capture waveforms, and patterns.
- Resets all properties to default values, including the *nidigital.Session.selected_function* property that is set to *DISCONNECT*, causing the I/O switches to open.
- Stops export of all external signals and events.
- Clears over-temperature and over-power conditions.

self_calibrate

nidigital.Session.**self_calibrate**()

Performs self-calibration on a digital pattern instrument.

self_test

nidigital.Session.**self_test**()

Returns self test results from a digital pattern instrument. This test requires several minutes to execute.

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

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

Self-Test Code	Description
0	Self test passed.
1	Self test failed.

send_software_edge_trigger

nidigital.Session.**send_software_edge_trigger**(*trigger*, *trigger_identifier*)

Forces a particular edge-based trigger to occur regardless of how the specified trigger is configured. You can use this method as a software override.

Parameters

- **trigger** (*nidigital.SoftwareTrigger*) – Trigger specifies the trigger you want to override.

Defined Values	
START	Overrides the Start trigger. You must specify an empty string in the trigger_identifier parameter.
CONDITIONA	Specifies to route a conditional jump trigger. You must specify a conditional jump trigger in the trigger_identifier parameter.

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

- **trigger_identifier** (*str*) – Trigger Identifier specifies the instance of the trigger you want to override. If trigger is specified as `NIDIGITAL_VAL_START_TRIGGER`, this parameter must be an empty string. If trigger is specified as `NIDIGITAL_VAL_CONDITIONAL_JUMP_TRIGGER`, allowed values are conditionalJumpTrigger0, conditionalJumpTrigger1, conditionalJumpTrigger2, and conditionalJumpTrigger3.

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

tdr

nidigital.Session.**tdr**(*apply_offsets=True*)

Measures propagation delays through cables, connectors, and load boards using Time-Domain Reflectometry (TDR). Ensure that the channels and pins you select are connected to an open circuit.

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

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

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

Example: `my_session.tdr()`

Parameters

apply_offsets (*bool*) – A Boolean that specifies whether to apply the measured

TDR offsets. If you need to adjust the measured offsets prior to applying, set this input to False, and call the *nidigital.Session.apply_tdr_offsets()* method to specify the adjusted TDR offsets values.

Return type
list of hightime.timedelta

Returns
Measured TDR offsets specified in seconds.

unload_all_patterns

nidigital.Session.**unload_all_patterns**(*unload_keep_alive_pattern=False*)

Unloads all patterns, source waveforms, and capture waveforms from a digital pattern instrument.

Parameters
unload_keep_alive_pattern (*bool*) – A Boolean that specifies whether to keep or unload the keep alive pattern.

unload_specifications

nidigital.Session.**unload_specifications**(*file_paths*)

Unloads the given specifications sheets present in the previously loaded specifications files that you select.

You must call *nidigital.Session.load_specifications_levels_and_timing()* to reload the files with updated specifications values. You must then call *nidigital.Session.apply_levels_and_timing()* in order to apply the levels and timing values that reference the updated specifications values.

Parameters
file_paths (*str or basic sequence of str*) – Absolute file path of one or more loaded specifications files.

unlock

nidigital.Session.**unlock**()

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

wait_until_done

nidigital.Session.**wait_until_done**(*timeout=hightime.timedelta(seconds=10.0)*)

Waits until the pattern burst has completed or the timeout has expired.

Parameters
timeout (*hightime.timedelta, datetime.timedelta, or float in seconds*) – Maximum time (in seconds) allowed for this method to complete. If this method does not complete within this time interval, this method returns an error.

write_sequencer_flag

nidigital.Session.**write_sequencer_flag**(*flag*, *value*)

Writes the state of a pattern sequencer flag. Use pattern sequencer flags to coordinate execution between the pattern sequencer and a runtime test program.

Parameters

- **flag** (*nidigital.SequencerFlag*) – The pattern sequencer flag to write.
  - *FLAG0* ("seqflag0"): Writes pattern sequencer flag 0.
  - *FLAG1* ("seqflag1"): Writes pattern sequencer flag 1.
  - *FLAG2* ("seqflag2"): Writes pattern sequencer flag 2.
  - *FLAG3* ("seqflag3"): Writes pattern sequencer flag 3.
- **value** (*bool*) – A Boolean that assigns a state to the pattern sequencer flag you specify.

write_sequencer_register

nidigital.Session.**write_sequencer_register**(*reg*, *value*)

Writes a value to a pattern sequencer register. Use pattern sequencer registers to pass numeric values between the pattern sequencer and a runtime test program.

Parameters

- **reg** (*nidigital.SequencerRegister*) – The sequencer register you want to write to.
  - *REGISTER0* ("reg0"): Writes sequencer register 0.
  - *REGISTER1* ("reg1"): Writes sequencer register 1.
  - *REGISTER2* ("reg2"): Writes sequencer register 2.
  - *REGISTER3* ("reg3"): Writes sequencer register 3.
  - *REGISTER4* ("reg4"): Writes sequencer register 4.
  - *REGISTER5* ("reg5"): Writes sequencer register 5.
  - *REGISTER6* ("reg6"): Writes sequencer register 6.
  - *REGISTER7* ("reg7"): Writes sequencer register 7.
  - *REGISTER8* ("reg8"): Writes sequencer register 8.
  - *REGISTER9* ("reg9"): Writes sequencer register 9.
  - *REGISTER10* ("reg10"): Writes sequencer register 10.
  - *REGISTER11* ("reg11"): Writes sequencer register 11.
  - *REGISTER12* ("reg12"): Writes sequencer register 12.
  - *REGISTER13* ("reg13"): Writes sequencer register 13.
  - *REGISTER14* ("reg14"): Writes sequencer register 14.
  - *REGISTER15* ("reg15"): Writes sequencer register 15.
- **value** (*int*) – The value you want to write to the register.

write_source_waveform_broadcast

nidigital.Session.**write_source_waveform_broadcast**(*waveform_name*, *waveform_data*)

Writes the same waveform data to all sites. Use this write method if you set the data_mapping parameter of the create source waveform method to *BROADCAST*.

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **waveform_data** (*list of int*) – 1D array of samples to use as source data to apply to all sites.

write_source_waveform_data_from_file_tdms

nidigital.Session.**write_source_waveform_data_from_file_tdms**(*waveform_name*, *waveform_file_path*)

Writes a source waveform based on the waveform data and configuration information the file contains.

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **waveform_file_path** (*str*) – Absolute file path to the load source waveform file (.tdms).

write_source_waveform_site_unique

nidigital.Session.**write_source_waveform_site_unique**(*waveform_name*, *waveform_data*)

Writes one waveform per site. Use this write method if you set the parameter of the create source waveform method to Site Unique.

Parameters

- **waveform_name** (*str*) – The name to assign to the waveform. Use the waveform_name with source_start opcode in your pattern.
- **waveform_data** (*{ int: basic sequence of unsigned int, int: basic sequence of unsigned int, ... }*) – Dictionary where each key is a site number and value is a collection of samples to use as source data

write_static

nidigital.Session.**write_static**(*state*)

Writes a static state to the specified pins. The selected pins remain in the specified state until the next pattern burst or call to this method. If there are uncommitted changes to levels or the termination mode, this method commits the changes to the pins. This method does not change the selected pin method. If you write a static state to a pin that does not have the Digital method selected, the new static state is stored by the instrument, and affects the state of the pin the next time you change the selected method to Digital.

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

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

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

Example: `my_session.write_static()`

Parameters

state (*nidigital.WriteStaticPinState*) – Parameter that specifies one of the following digital states to assign to the pin.

- *ZERO*: Specifies to drive low.
- *ONE*: Specifies to drive high.
- *X*: Specifies to not drive.

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

Properties

active_load_ioh

`nidigital.Session.`**`active_load_ioh`**

Specifies the current that the DUT sources to the active load while outputting a voltage above VCOM.

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

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

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

Example: `my_session.active_load_ioh`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_ACTIVE_LOAD_IOH**

active_load_iol

nidigital.Session.**active_load_iol**

Specifies the current that the DUT sinks from the active load while outputting a voltage below VCOM.

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

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

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

Example: `my_session.active_load_iol`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_ACTIVE_LOAD_IOL**

---

active_load_vcom

nidigital.Session.**active_load_vcom**

Specifies the voltage level at which the active load circuit switches between sourcing current and sinking current.

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

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

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

Example: `my_session.active_load_vcom`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_ACTIVE_LOAD_VCOM**

cache

`nidigital.Session.`**`cache`**

Specifies whether to cache the value of properties. When caching is enabled, the instrument driver keeps track of the current instrument settings and avoids sending redundant commands to the instrument. This significantly increases execution speed. Caching is always enabled in the driver, regardless of the value of this property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_CACHE**

channel_count

`nidigital.Session.`**`channel_count`**

Returns the number of channels that the specific digital pattern instrument driver supports.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_CHANNEL_COUNT**

clock_generator_frequency

nidigital.Session.**clock_generator_frequency**

Specifies the frequency for the clock generator.

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

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

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

Example: `my_session.clock_generator_frequency`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_CLOCK_GENERATOR_FREQUENCY**

---

clock_generator_is_running

nidigital.Session.**clock_generator_is_running**

Indicates whether the clock generator is running.

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

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

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

Example: `my_session.clock_generator_is_running`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read only
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_CLOCK_GENERATOR_IS_RUNNING**

conditional_jump_trigger_terminal_name

nidigital.Session.**conditional_jump_trigger_terminal_name**

Specifies the terminal name from which the exported conditional jump trigger signal may be routed to other instruments through the PXI trigger bus. You can use this signal to trigger other instruments when the conditional jump trigger instance asserts on the digital pattern instrument.

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

Example: `my_session.conditional_jump_triggers[ ... ].conditional_jump_trigger_terminal_name`

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

Example: `my_session.conditional_jump_trigger_terminal_name`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_CONDITIONAL_JUMP_TRIGGER_TERMINAL_NAME**

conditional_jump_trigger_type

nidigital.Session.**conditional_jump_trigger_type**

Disables the conditional jump trigger or configures it for either hardware triggering or software triggering. The default value is *NONE*.

Valid Values:	
NONE	Disables the conditional jump trigger.
DIGITAL_EDGE	Configures the conditional jump trigger for hardware triggering.
SOFTWARE	Configures the conditional jump trigger for software triggering.

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

Example: `my_session.conditional_jump_triggers[ ... ].conditional_jump_trigger_type`

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

Example: `my_session.conditional_jump_trigger_type`

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_CONDITIONAL_JUMP_TRIGGER_TYPE**

cycle_number_history_ram_trigger_cycle_number

`nidigital.Session.`**`cycle_number_history_ram_trigger_cycle_number`**

Specifies the cycle number on which History RAM starts acquiring pattern information when configured for a cycle number trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_CYCLE_NUMBER_HISTORY_RAM_TRIGGER_CYCLE_NUMBER**

digital_edge_conditional_jump_trigger_edge

nidigital.Session.**digital_edge_conditional_jump_trigger_edge**

Configures the active edge of the incoming trigger signal for the conditional jump trigger instance. The default value is *RISING*.

Valid Values:	
RISING	Specifies the signal transition from low level to high level.
FALLING	Specifies the signal transition from high level to low level.

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

Example: `my_session.conditional_jump_triggers[ ... ].digital_edge_conditional_jump_trigger_edge`

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

Example: `my_session.digital_edge_conditional_jump_trigger_edge`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.DigitalEdge
Permissions	read-write
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_CONDITIONAL_JUMP_TRIGGER_EDGE**

digital_edge_conditional_jump_trigger_source

nidigital.Session.**digital_edge_conditional_jump_trigger_source**

Configures the digital trigger source terminal for a conditional jump trigger instance. The PXIe-6570/6571 supports triggering through the PXI trigger bus. You can specify source terminals in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The source terminal can also be a terminal from another device, in which case the NI-Digital Pattern Driver automatically finds a route (if one is available) from that terminal to the input terminal (going through a physical PXI backplane trigger line). For example, you can set the source terminal on Dev1 to be /Dev2/ConditionalJumpTrigger0. The default value is VI_NULL.

Valid Values:
String identifier to any valid terminal name

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

Example: `my_session.conditional_jump_triggers[ ... ].digital_edge_conditional_jump_trigger_source`

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

Example: `my_session.digital_edge_conditional_jump_trigger_source`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_CONDITIONAL_JUMP_TRIGGER_SOURCE**

digital_edge_rio_trigger_edge

`nidigital.Session.`**`digital_edge_rio_trigger_edge`**

Configures the active edge of the incoming trigger signal for the RIO trigger instance. The default value is *RISING*.

Valid Values:	
RISING	Specifies the signal transition from low level to high level.
FALLING	Specifies the signal transition from high level to low level.

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

Example: `my_session.rio_triggers[ ... ].digital_edge_rio_trigger_edge`

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

Example: `my_session.digital_edge_rio_trigger_edge`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.DigitalEdge
Permissions	read-write
Repeated Capabilities	rio_triggers

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_RIO_TRIGGER_EDGE**

digital_edge_rio_trigger_source

`nidigital.Session.`**`digital_edge_rio_trigger_source`**

Configures the digital trigger source terminal for a RIO trigger instance. The PXIe-6570/6571 supports triggering through the PXI trigger bus. You can specify source terminals in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The source terminal can also be a terminal from another device, in which case the NI-Digital Pattern Driver automatically finds a route (if one is available) from that terminal to the input terminal (going through a physical PXI backplane trigger line). For example, you can set the source terminal on Dev1 to be /Dev2/RIOTrigger0. The default value is VI_NULL.

Valid Values:
String identifier to any valid terminal name

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

Example: `my_session.rio_triggers[ ... ].digital_edge_rio_trigger_source`

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

Example: `my_session.digital_edge_rio_trigger_source`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	rio_triggers

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_RIO_TRIGGER_SOURCE**

digital_edge_start_trigger_edge

nidigital.Session.**digital_edge_start_trigger_edge**

Specifies the active edge for the Start trigger. This property is used when the *nidigital.Session.start_trigger_type* property is set to Digital Edge.

Defined Values:	
RISING	Asserts the trigger when the signal transitions from low level to high level.
FALLING	Asserts the trigger when the signal transitions from high level to low level.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_START_TRIGGER_EDGE**

digital_edge_start_trigger_source

nidigital.Session.**digital_edge_start_trigger_source**

Specifies the source terminal for the Start trigger. This property is used when the *nidigital.Session.start_trigger_type* property is set to Digital Edge. You can specify source terminals in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0. The source terminal can also be a terminal from another device, in which case the NI-Digital Pattern Driver automatically finds a route (if one is available) from that terminal to the input terminal (going through a physical PXI backplane trigger line). For example, you can set the source terminal on Dev1 to be /Dev2/StartTrigger.

Defined Values:	
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_DIGITAL_EDGE_START_TRIGGER_SOURCE**

driver_setup

`nidigital.Session.`**`driver_setup`**

This property returns initial values for NI-Digital Pattern Driver properties as a string.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_DRIVER_SETUP**

exported_conditional_jump_trigger_output_terminal

`nidigital.Session.`**`exported_conditional_jump_trigger_output_terminal`**

Specifies the terminal to output the exported signal of the specified instance of the conditional jump trigger. The default value is VI_NULL.

Valid Values:	
VI_NULL (“”)	Returns an empty string
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

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

Example: `my_session.conditional_jump_triggers[ ... ].exported_conditional_jump_trigger_output_terminal`

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

Example: `my_session.exported_conditional_jump_trigger_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	conditional_jump_triggers

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

- C Attribute: **NIDIGITAL_ATTR_EXPORTED_CONDITIONAL_JUMP_TRIGGER_OUTPUT_TERMINAL**

exported_pattern_opcode_event_output_terminal

`nidigital.Session.`**`exported_pattern_opcode_event_output_terminal`**

Specifies the destination terminal for exporting the Pattern Opcode Event. Terminals can be specified in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Defined Values:	
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

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

Example: `my_session.pattern_opcode_events[ ... ].exported_pattern_opcode_event_output_terminal`

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

Example: `my_session.exported_pattern_opcode_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	pattern_opcode_events

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

- C Attribute: **NIDIGITAL_ATTR_EXPORTED_PATTERN_OPCODE_EVENT_OUTPUT_TERMINAL**

exported_rio_event_output_terminal

`nidigital.Session.`**`exported_rio_event_output_terminal`**

Specifies the destination terminal for exporting the RIO Event. Terminals can be specified in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Defined Values:	
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

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

Example: `my_session.rio_events[ ... ].exported_rio_event_output_terminal`

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

Example: `my_session.exported_rio_event_output_terminal`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	rio_events

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

- C Attribute: **NIDIGITAL_ATTR_EXPORTED_RIO_EVENT_OUTPUT_TERMINAL**

exported_start_trigger_output_terminal

nidigital.Session.**exported_start_trigger_output_terminal**

Specifies the destination terminal for exporting the Start trigger. Terminals can be specified in one of two ways. If the digital pattern instrument is named Dev1 and your terminal is PXI_Trig0, you can specify the terminal with the fully qualified terminal name, /Dev1/PXI_Trig0, or with the shortened terminal name, PXI_Trig0.

Defined Values:	
Do not export signal	The signal is not exported.
PXI_Trig0	PXI trigger line 0
PXI_Trig1	PXI trigger line 1
PXI_Trig2	PXI trigger line 2
PXI_Trig3	PXI trigger line 3
PXI_Trig4	PXI trigger line 4
PXI_Trig5	PXI trigger line 5
PXI_Trig6	PXI trigger line 6
PXI_Trig7	PXI trigger line 7

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_EXPORTED_START_TRIGGER_OUTPUT_TERMINAL**

frequency_counter_hysteresis_enabled

nidigital.Session.**frequency_counter_hysteresis_enabled**

Specifies whether hysteresis is enabled for the frequency counters of the digital pattern instrument.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_FREQUENCY_COUNTER_HYSTERESIS_ENABLED**

frequency_counter_measurement_mode

nidigital.Session.**frequency_counter_measurement_mode**

Determines how the frequency counters of the digital pattern instrument make measurements.

Valid Values:	
BANK	Each discrete frequency counter is mapped to specific channels and makes frequency measurements from only those channels. Use banked mode when you need access to the full measure frequency range of the instrument. **Note:** If you request frequency measurements from multiple channels within the same bank, the measurements are made in series for the channels in that bank.
PARA	All discrete frequency counters make frequency measurements from all channels in parallel with one another. Use parallel mode to increase the speed of frequency measurements if you do not need access to the full measure frequency range of the instrument; in parallel mode, you can also add *nidigital.Session.frequency_counter_hysteresis_enabled* to reduce measurement noise.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_FREQUENCY_COUNTER_MEASUREMENT_MODE**

frequency_counter_measurement_time

nidigital.Session.**frequency_counter_measurement_time**

Specifies the measurement time for the frequency counter.

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

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

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

Example: `my_session.frequency_counter_measurement_time`

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_FREQUENCY_COUNTER_MEASUREMENT_TIME**

group_capabilities

`nidigital.Session.`**`group_capabilities`**

Returns a string that contains a comma-separated list of class-extension groups that the driver implements.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_GROUP_CAPABILITIES**

halt_on_keep_alive_opcode

`nidigital.Session.`**`halt_on_keep_alive_opcode`**

Specifies whether keep_alive opcodes should behave like halt opcodes.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_HALT_ON_KEEP_ALIVE_OPCODE**

history_ram_buffer_size_per_site

nidigital.Session.**history_ram_buffer_size_per_site**

Specifies the size, in samples, of the host memory buffer. The default value is 32000.

Valid Values:
0-INT64_MAX

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_HISTORY_RAM_BUFFER_SIZE_PER_SITE**

history_ram_cycles_to_acquire

nidigital.Session.**history_ram_cycles_to_acquire**

Configures which cycles History RAM acquires after the trigger conditions are met. If you configure History RAM to only acquire failed cycles, you must set the pretrigger samples for History RAM to 0.

Defined Values:	
FAILED	Only acquires cycles that fail a compare after the triggering conditions are met.
ALL	Acquires all cycles after the triggering conditions are met.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_HISTORY_RAM_CYCLES_TO_ACQUIRE**

history_ram_max_samples_to_acquire_per_site

nidigital.Session.**history_ram_max_samples_to_acquire_per_site**

Specifies the maximum number of History RAM samples to acquire per site. If the property is set to -1, it will acquire until the History RAM buffer is full.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_HISTORY_RAM_MAX_SAMPLES_TO_ACQUIRE_PER_SITE**

history_ram_number_of_samples_is_finite

nidigital.Session.**history_ram_number_of_samples_is_finite**

Specifies whether the instrument acquires a finite number of History Ram samples or acquires continuously. The maximum number of samples that will be acquired when this property is set to True is determined by the instrument History RAM depth specification and the History RAM Max Samples to Acquire Per Site property. The default value is True.

Valid Values:	
True	Specifies that History RAM results will not stream into the host buffer until a History RAM fetch API is called.
False	Specifies that History RAM results will automatically start streaming into a host buffer after a pattern is burst and the History RAM has triggered.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_HISTORY_RAM_NUMBER_OF_SAMPLES_IS_FINITE**

history_ram_pretrigger_samples

nidigital.Session.**history_ram_pretrigger_samples**

Specifies the number of samples to acquire before the trigger conditions are met. If you configure History RAM to only acquire failed cycles, you must set the pretrigger samples for History RAM to 0.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_HISTORY_RAM_PRETRIGGER_SAMPLES**

history_ram_trigger_type

nidigital.Session.**history_ram_trigger_type**

Specifies the type of trigger condition on which History RAM starts acquiring pattern information.

Defined Values:	
FIRST_FAI	Starts acquiring pattern information in History RAM on the first failed cycle in a pattern burst.
CYCLE_NUM	Starts acquiring pattern information in History RAM starting from a specified cycle number.
PATTERN_L	Starts acquiring pattern information in History RAM starting from a specified pattern label, augmented by vector and cycle offsets.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_HISTORY_RAM_TRIGGER_TYPE**

instrument_firmware_revision

nidigital.Session.**instrument_firmware_revision**

Returns a string that contains the firmware revision information for the digital pattern instrument.

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

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

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

Example: `my_session.instrument_firmware_revision`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	instruments

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

- C Attribute: **NIDIGITAL_ATTR_INSTRUMENT_FIRMWARE_REVISION**

---

instrument_manufacturer

nidigital.Session.**instrument_manufacturer**

Returns a string (“National Instruments”) that contains the name of the manufacturer of the digital pattern instrument.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_INSTRUMENT_MANUFACTURER**

---

instrument_model

nidigital.Session.**instrument_model**

Returns a string that contains the model number or name of the digital pattern instrument.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_INSTRUMENT_MODEL**

interchange_check

nidigital.Session.**interchange_check**

This property is not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_INTERCHANGE_CHECK**

io_resource_descriptor

nidigital.Session.**io_resource_descriptor**

Returns a string that contains the resource descriptor that the NI-Digital Pattern Driver uses to identify the digital pattern instrument.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_IO_RESOURCE_DESCRIPTOR**

is_keep_alive_active

`nidigital.Session.`**`is_keep_alive_active`**

Returns True if the digital pattern instrument is driving the keep alive pattern.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_IS_KEEP_ALIVE_ACTIVE**

logical_name

`nidigital.Session.`**`logical_name`**

Returns a string containing the logical name that you specified when opening the current IVI session. This property is not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_LOGICAL_NAME**

mask_compare

nidigital.Session.**mask_compare**

Specifies whether the pattern comparisons are masked or not. When set to True for a specified pin, failures on that pin will be masked.

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

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

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

Example: `my_session.mask_compare`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_MASK_COMPARE**

---

pattern_label_history_ram_trigger_cycle_offset

nidigital.Session.**pattern_label_history_ram_trigger_cycle_offset**

Specifies the number of cycles that follow the specified pattern label and vector offset, after which History RAM will start acquiring pattern information when configured for a pattern label trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_PATTERN_LABEL_HISTORY_RAM_TRIGGER_CYCLE_OFFSET**

---

pattern_label_history_ram_trigger_label

nidigital.Session.**pattern_label_history_ram_trigger_label**

Specifies the pattern label, augmented by the vector and cycle offset, to determine the point where History RAM will start acquiring pattern information when configured for a pattern label trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_PATTERN_LABEL_HISTORY_RAM_TRIGGER_LABEL**

pattern_label_history_ram_trigger_vector_offset

nidigital.Session.**pattern_label_history_ram_trigger_vector_offset**

Specifies the number of vectors that follow the specified pattern label, after which History RAM will start acquiring pattern information when configured for a pattern label trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_PATTERN_LABEL_HISTORY_RAM_TRIGGER_VECTOR_OFFSET**

pattern_opcode_event_terminal_name

nidigital.Session.**pattern_opcode_event_terminal_name**

Specifies the terminal name for the output trigger signal of the specified instance of a Pattern Opcode Event. You can use this terminal name as an input signal source for another trigger.

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

Example: `my_session.pattern_opcode_events[ ... ].pattern_opcode_event_terminal_name`

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

Example: `my_session.pattern_opcode_event_terminal_name`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	pattern_opcode_events

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

- C Attribute: **NIDIGITAL_ATTR_PATTERN_OPCODE_EVENT_TERMINAL_NAME**

ppmu_allow_extended_voltage_range

`nidigital.Session.`**`ppmu_allow_extended_voltage_range`**

Enables the instrument to operate in additional voltage ranges where instrument specifications may differ from standard ranges. When set to True, this property enables extended voltage range operation. Review specification deviations for application suitability before using this property. NI recommends setting this property to False when not using the extended voltage range to avoid unintentional use of this range. The extended voltage range is supported only for PPMU, with the output method set to DC Voltage. A voltage glitch may occur when you change the PPMU output voltage from a standard range to the extended voltage range, or vice-versa, while the PPMU is sourcing. NI recommends temporarily changing the *nidigital.Session.selected_function* property to Off before sourcing a voltage level that requires a range change.

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

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

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

Example: `my_session.ppmu_allow_extended_voltage_range`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_ALLOW_EXTENDED_VOLTAGE_RANGE**

ppmu_aperture_time

nidigital.Session.**ppmu_aperture_time**

Specifies the measurement aperture time for the PPMU. The *nidigital.Session.ppmu_aperture_time_units* property sets the units of the PPMU aperture time.

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

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

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

Example: `my_session.ppmu_aperture_time`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_APERTURE_TIME**

ppmu_aperture_time_units

nidigital.Session.**ppmu_aperture_time_units**

Specifies the units of the measurement aperture time for the PPMU.

Defined Values:	
SECONDS	Specifies the aperture time in seconds.

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

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

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

Example: `my_session.ppmu_aperture_time_units`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.PPMUApertureTimeUnits
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_APERTURE_TIME_UNITS**

ppmu_current_level

nidigital.Session.**ppmu_current_level**

Specifies the current level, in amps, that the PPMU forces to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Current. Specify valid values for the current level using the `nidigital.Session.PPMU_ConfigureCurrentLevelRange()` method.

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

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

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

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

Example: `my_session.ppmu_current_level`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LEVEL**

ppmu_current_level_range

nidigital.Session.**ppmu_current_level_range**

Specifies the range of valid values for the current level, in amps, that the PPMU forces to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Current.

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

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

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

Example: `my_session.ppmu_current_level_range`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LEVEL_RANGE**

---

ppmu_current_limit

nidigital.Session.**ppmu_current_limit**

Specifies the current limit, in amps, that the output cannot exceed while the PPMU forces voltage to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Voltage. The PXIe-6570/6571 does not support the *nidigital.Session.ppmu_current_limit* property and only allows configuration of the *nidigital.Session.ppmu_current_limit_range* property.

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

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

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

Example: `my_session.ppmu_current_limit`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LIMIT**

ppmu_current_limit_behavior

nidigital.Session.**ppmu_current_limit_behavior**

Specifies how the output should behave when the current limit is reached.

Defined Values:	
REGULATE	Controls output current so that it does not exceed the current limit. Power continues to generate even if the current limit is reached.

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

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

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

Example: `my_session.ppmu_current_limit_behavior`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.PPMUCurrentLimitBehavior
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LIMIT_BEHAVIOR**

ppmu_current_limit_range

nidigital.Session.**ppmu_current_limit_range**

Specifies the valid range, in amps, to which the current limit can be set while the PPMU forces voltage to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Voltage.

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

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

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

Example: `my_session.ppmu_current_limit_range`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LIMIT_RANGE**

---

ppmu_current_limit_supported

nidigital.Session.**ppmu_current_limit_supported**

Returns whether the device supports configuration of a current limit when you set the *nidigital.Session.ppmu_output_function* property to DC Voltage.

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

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

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

Example: `my_session.ppmu_current_limit_supported`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read only
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_CURRENT_LIMIT_SUPPORTED**

ppmu_output_function

nidigital.Session.**ppmu_output_function**

Specifies whether the PPMU forces voltage or current to the DUT.

Defined Values:	
VOLTAGE	Specifies the output method to DC Voltage.
CURRENT	Specifies the output method to DC Current.

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

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

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

Example: `my_session.ppmu_output_function`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.PPMUOutputFunction
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_OUTPUT_FUNCTION**

ppmu_voltage_level

nidigital.Session.**ppmu_voltage_level**

Specifies the voltage level, in volts, that the PPMU forces to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Voltage.

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

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

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

Example: `my_session.ppmu_voltage_level`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_VOLTAGE_LEVEL**

ppmu_voltage_limit_high

`nidigital.Session.`**`ppmu_voltage_limit_high`**

Specifies the maximum voltage limit, or high clamp voltage (V_{CH}), in volts, at the pin when the PPMU forces current to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Current.

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

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

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

Example: `my_session.ppmu_voltage_limit_high`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_VOLTAGE_LIMIT_HIGH**

ppmu_voltage_limit_low

nidigital.Session.**ppmu_voltage_limit_low**

Specifies the minimum voltage limit, or low clamp voltage (V_{CL}), in volts, at the pin when the PPMU forces current to the DUT. This property is applicable only when you set the *nidigital.Session.ppmu_output_function* property to DC Current.

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

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

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

Example: `my_session.ppmu_voltage_limit_low`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_PPMU_VOLTAGE_LIMIT_LOW**

---

query_instrument_status

nidigital.Session.**query_instrument_status**

Specifies whether the NI-Digital Pattern Driver queries the digital pattern instrument status after each operation. The instrument status is always queried, regardless of the property setting.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_QUERY_INSTRUMENT_STATUS**

---

range_check

nidigital.Session.**range_check**

Checks the range and validates parameter and property values you pass to NI-Digital Pattern Driver methods. Ranges are always checked, regardless of the property setting.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_RANGE_CHECK**

record_coercions

nidigital.Session.**record_coercions**

Specifies whether the IVI engine keeps a list of the value coercions it makes for integer and real type properties. Enabling record value coercions is not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_RECORD_COERCIONS**

rio_event_terminal_name

nidigital.Session.**rio_event_terminal_name**

Specifies the terminal name for the output signal of the specified instance of a RIO Event. You can use this terminal name as an input signal source for another trigger.

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

Example: `my_session.rio_events[ ... ].rio_event_terminal_name`

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

Example: `my_session.rio_event_terminal_name`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	rio_events

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

- C Attribute: **NIDIGITAL_ATTR_RIO_EVENT_TERMINAL_NAME**

rio_trigger_terminal_name

`nidigital.Session.`**`rio_trigger_terminal_name`**

Specifies the terminal name from which the exported RIO trigger signal may be routed to other instruments through the PXI trigger bus. You can use this signal to trigger other instruments when the RIO trigger instance asserts on the digital pattern instrument.

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

Example: `my_session.rio_triggers[ ... ].rio_trigger_terminal_name`

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

Example: `my_session.rio_trigger_terminal_name`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	rio_triggers

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

- C Attribute: **NIDIGITAL_ATTR_RIO_TRIGGER_TERMINAL_NAME**

rio_trigger_type

nidigital.Session.**rio_trigger_type**

Disables the rio trigger or configures it for hardware triggering. The default value is *NONE*.

Valid Values:	
NONE	Disables the conditional jump trigger.
DIGITAL_EDGE	Configures the conditional jump trigger for hardware triggering.

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

Example: `my_session.rio_triggers[ ... ].rio_trigger_type`

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

Example: `my_session.rio_trigger_type`

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_RIO_TRIGGER_TYPE**

selected_function

nidigital.Session.**selected_function**

Caution: In the Disconnect state, some I/O protection and sensing circuitry remains exposed. Do not subject the instrument to voltage beyond its operating range.

Specifies whether digital pattern instrument channels are controlled by the pattern sequencer or PPMU, disconnected, or off.

Defined Values:	
DIGI	The pin is connected to the driver, comparator, and active load methods. The PPMU is not sourcing, but can make voltage measurements. The state of the digital pin driver when you change the *nidigital.Session.selected_function* to Digital is determined by the most recent call to the *nidigital.Session.write_static()* method or the last vector of the most recently executed pattern burst, whichever happened last. Use the *nidigital.Session.write_static()* method to control the state of the digital pin driver through software. Use the *nidigital.Session.burst_pattern()* method to control the state of the digital pin driver through a pattern. Set the **selectDigitalFunction** parameter of the *nidigital.Session.burst_pattern()* method to True to automatically switch the *nidigital.Session.selected_function* of the pins in the pattern burst to *DIGITAL*.
PPMU	The pin is connected to the PPMU. The driver, comparator, and active load are off while this method is selected. Call the *nidigital.Session.ppmu_source()* method to source a voltage or current. The *nidigital.Session.ppmu_source()* method automatically switches the *nidigital.Session.selected_function* to the PPMU state and starts sourcing from the PPMU. Changing the *nidigital.Session.selected_function* to *DISCONNECT*, *OFF*, or *DIGITAL* causes the PPMU to stop sourcing. If you set the *nidigital.Session.selected_function* property to PPMU, the PPMU is initially not sourcing.
OFF	The pin is electrically connected, and the PPMU and digital pin driver are off while this method is selected.
DISC	The pin is electrically disconnected from instrument methods. Selecting this method causes the PPMU to stop sourcing prior to disconnecting the pin.

Note: You can make PPMU voltage measurements using the *nidigital.Session.ppmu_measure()* method from within any *nidigital.Session.selected_function*.

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

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

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

Example: `my_session.selected_function`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.SelectedFunction
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_SELECTED_FUNCTION**

sequencer_flag_terminal_name

nidigital.Session.**sequencer_flag_terminal_name**

Specifies the terminal name for the output trigger signal of the Sequencer Flags trigger. You can use this terminal name as an input signal source for another trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SEQUENCER_FLAG_TERMINAL_NAME**

serial_number

nidigital.Session.**serial_number**

Returns the serial number of the device.

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

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

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

Example: `my_session.serial_number`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	instruments

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

- C Attribute: **NIDIGITAL_ATTR_SERIAL_NUMBER**

simulate

nidigital.Session.**simulate**

Simulates I/O operations. After you open a session, you cannot change the simulation state. Use the nidigital.Session.__init__() method to enable simulation.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SIMULATE**

specific_driver_class_spec_major_version

nidigital.Session.**specific_driver_class_spec_major_version**

Returns the major version number of the class specification with which NI-Digital is compliant. This property is not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MAJOR_VERSION**

specific_driver_class_spec_minor_version

nidigital.Session.**specific_driver_class_spec_minor_version**

Returns the minor version number of the class specification with which NI-Digital is compliant. This property is not supported.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	int
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_CLASS_SPEC_MINOR_VERSION**

specific_driver_description

nidigital.Session.**specific_driver_description**

Returns a string that contains a brief description of the NI-Digital Pattern driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_DESCRIPTION**

specific_driver_prefix

nidigital.Session.**specific_driver_prefix**

Returns a string that contains the prefix for the NI-Digital Pattern driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_PREFIX**

specific_driver_revision

nidigital.Session.**specific_driver_revision**

Returns a string that contains additional version information about the NI-Digital Pattern Driver. For example, the driver can return Driver: NI-Digital 16.0 as the value of this property.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_REVISION**

specific_driver_vendor

nidigital.Session.**specific_driver_vendor**

Returns a string ("National Instruments") that contains the name of the vendor that supplies the NI-Digital Pattern Driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SPECIFIC_DRIVER_VENDOR**

start_label

nidigital.Session.**start_label**

Specifies the pattern name or exported pattern label from which to start bursting the pattern.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_START_LABEL**

start_trigger_terminal_name

nidigital.Session.**start_trigger_terminal_name**

Specifies the terminal name for the output trigger signal of the Start trigger. You can use this terminal name as an input signal source for another trigger.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_START_TRIGGER_TERMINAL_NAME**

start_trigger_type

nidigital.Session.**start_trigger_type**

Specifies the Start trigger type. The digital pattern instrument waits for this trigger after you call the `nidigital.Session.init()` method or the *nidigital.Session.burst_pattern()* method, and does not burst a pattern until this trigger is received.

Defined Values:	
NONE	Disables the Start trigger. Pattern bursting starts immediately after you call the `nidigital.Session.init()` method or the *nidigital.Session.burst_pattern()* method.
DIGI1	Pattern bursting does not start until the digital pattern instrument detects a digital edge.
SOFTW	Pattern bursting does not start until the digital pattern instrument receives a software Start trigger. Create a software Start trigger by calling the *nidigital.Session.send_software_edge_trigger()* method and selecting start trigger in the **trigger** parameter.Related information: SendSoftwareEdgeTrigger method.

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

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_START_TRIGGER_TYPE**

supported_instrument_models

nidigital.Session.**supported_instrument_models**

Returns a comma delimited string that contains the supported digital pattern instrument models for the specific driver.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	str
Permissions	read only
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_SUPPORTED_INSTRUMENT_MODELS**

tdr_endpoint_termination

nidigital.Session.**tdr_endpoint_termination**

Specifies whether TDR Channels are connected to an open circuit or a short to ground.

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_TDR_ENDPOINT_TERMINATION**

tdr_offset

nidigital.Session.**tdr_offset**

Specifies the TDR Offset.

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

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

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

Example: `my_session.tdr_offset`

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_TDR_OFFSET**

---

termination_mode

nidigital.Session.**termination_mode**

Specifies the behavior of the pin during non-drive cycles.

Defined Values:	
ACTI	Specifies that, for non-drive pin states (L, H, X, V, M, E), the active load is connected and the instrument sources or sinks a defined amount of current to load the DUT. The amount of current sourced by the instrument and therefore sunk by the DUT is specified by IOL. The amount of current sunk by the instrument and therefore sourced by the DUT is specified by IOH. The voltage at which the instrument changes between sourcing and sinking is specified by VCOM.
VTER	Specifies that, for non-drive pin states (L, H, X, V, M, E), the pin driver terminates the pin to the configured VTERM voltage through a 50 impedance. VTERM is adjustable to allow for the pin to terminate at a set level. This is useful for instruments that might operate incorrectly if an instrument pin is unterminated and is allowed to float to any voltage level within the instrument voltage range. To address this issue, enable VTERM by configuring the VTERM pin level to the desired voltage and selecting the VTERM termination mode. Setting VTERM to 0 V and selecting the VTERM termination mode has the effect of connecting a 50 termination to ground, which provides an effective 50 impedance for the pin. This can be useful for improving signal integrity of certain DUTs by reducing reflections while the DUT drives the pin.
HIGH	Specifies that, for non-drive pin states (L, H, X, V, M, E), the pin driver is put in a high-impedance state and the active load is disabled.

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

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

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

Example: `my_session.termination_mode`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	enums.TerminationMode
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_TERMINATION_MODE**

timing_absolute_delay

nidigital.Session.**timing_absolute_delay**

Specifies a timing delay, measured in seconds, and applies the delay to the digital pattern instrument in addition to TDR and calibration adjustments. If the *nidigital.Session.timing_absolute_delay_enabled* property is set to True, this value is the intermodule skew measured by NI-TClk. You can modify this value to override the timing delay and align the I/O timing of this instrument with another instrument that shares the same reference clock. If the *nidigital.Session.timing_absolute_delay_enabled* property is False, this property will return 0.0. Changing the *nidigital.Session.timing_absolute_delay_enabled* property from False to True will set the *nidigital.Session.timing_absolute_delay* value back to your previously set value.

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

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

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

Example: `my_session.timing_absolute_delay`

The following table lists the characteristics of this property.

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

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

- C Attribute: **NIDIGITAL_ATTR_TIMING_ABSOLUTE_DELAY**

---

timing_absolute_delay_enabled

nidigital.Session.**timing_absolute_delay_enabled**

Specifies whether the *nidigital.Session.timing_absolute_delay* property should be applied to adjust the digital pattern instrument timing reference relative to other instruments in the system. Do not use this feature with digital pattern instruments in a Semiconductor Test System (STS). Timing absolute delay conflicts with the adjustment performed during STS timing calibration. When set to True, the digital pattern instrument automatically adjusts the timing absolute delay to correct the instrument timing reference relative to other instruments in the system for better timing alignment among synchronized instruments.

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	bool
Permissions	read-write
Repeated Capabilities	None

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

- C Attribute: **NIDIGITAL_ATTR_TIMING_ABSOLUTE_DELAY_ENABLED**

vih

nidigital.Session.**vih**

Specifies the voltage that the digital pattern instrument will apply to the input of the DUT when the test instrument drives a logic high (1).

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

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

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

Example: `my_session.vih`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VIH**

vil

nidigital.Session.**vil**

Specifies the voltage that the digital pattern instrument will apply to the input of the DUT when the test instrument drives a logic low (0).

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

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

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

Example: `my_session.vil`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VIL**

voh

`nidigital.Session.`**voh**

Specifies the output voltage from the DUT above which the comparator on the digital pattern test instrument interprets a logic high (H).

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

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

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

Example: `my_session.voh`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VOH**

vol

nidigital.Session.**vol**

Specifies the output voltage from the DUT below which the comparator on the digital pattern test instrument interprets a logic low (L).

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

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

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

Example: `my_session.vol`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VOL**

---

vterm

nidigital.Session.**vterm**

Specifies the termination voltage the digital pattern instrument applies during non-drive cycles when the termination mode is set to V_{term}. The instrument applies the termination voltage through a 50 parallel termination resistance.

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

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

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

Example: `my_session.vterm`

The following table lists the characteristics of this property.

Characteristic	Value
Datatype	float
Permissions	read-write
Repeated Capabilities	channels, pins

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

- C Attribute: **NIDIGITAL_ATTR_VTERM**

NI-TClk Support

`nidigital.Session.tclk`

This is used to get and set NI-TClk attributes on the session.

See also:

See `nitclk.SessionReference` for a complete list of attributes.

Session

- *create_capture_waveform_serial*
- *create_source_waveform_from_file_tdms*
- *create_source_waveform_parallel*
- *create_source_waveform_serial*
- *create_time_set*
- *delete_all_time_sets*
- *disable_sites*
- *enable_sites*
- *fetch_capture_waveform*
- *fetch_history_ram_cycle_information*
- *frequency_counter_measure_frequency*
- *get_channel_names*
- *get_fail_count*
- *get_history_ram_sample_count*
- *get_pattern_pin_names*
- *get_pin_results_pin_information*
- *get_site_pass_fail*
- *get_time_set_drive_format*
- *get_time_set_edge*
- *get_time_set_edge_multiplier*
- *get_time_set_period*
- *initiate*
- *is_done*
- *is_site_enabled*
- *load_pattern*
- *load_pin_map*
- *load_specifications_levels_and_timing*
- *lock*
- *ppmu_measure*
- *ppmu_source*
- *read_sequencer_flag*
- *read_sequencer_register*
- *read_static*
- *reset*
- *reset_device*

- *exported_start_trigger_output_terminal*
- *frequency_counter_hysteresis_enabled*
- *frequency_counter_measurement_mode*
- *frequency_counter_measurement_time*
- *group_capabilities*
- *halt_on_keep_alive_opcode*
- *history_ram_buffer_size_per_site*
- *history_ram_cycles_to_acquire*
- *history_ram_max_samples_to_acquire_per_site*
- *history_ram_number_of_samples_is_finite*
- *history_ram_pretrigger_samples*
- *history_ram_trigger_type*
- *instrument_firmware_revision*
- *instrument_manufacturer*
- *instrument_model*
- *interchange_check*
- *io_resource_descriptor*
- *is_keep_alive_active*
- *logical_name*
- *mask_compare*
- *pattern_label_history_ram_trigger_cycle_offset*
- *pattern_label_history_ram_trigger_label*
- *pattern_label_history_ram_trigger_vector_offset*
- *pattern_opcode_event_terminal_name*
- *ppmu_allow_extended_voltage_range*
- *ppmu_aperture_time*
- *ppmu_aperture_time_units*
- *ppmu_current_level*
- *ppmu_current_level_range*
- *ppmu_current_limit*
- *ppmu_current_limit_behavior*
- *ppmu_current_limit_range*
- *ppmu_current_limit_supported*
- *ppmu_output_function*
- *ppmu_voltage_level*

Repeated Capabilities

Repeated capabilities attributes are used to set the *channel_string* parameter to the underlying driver function call. This can be the actual function based on the `Session` method being called, or it can be the appropriate Get/Set Attribute function, such as `niDigital_SetAttributeViInt32()`.

Repeated capabilities attributes use the indexing operator `[]` to indicate the repeated capabilities. The parameter can be a string, list, tuple, or slice (range). Each element of those can be a string or an integer. If it is a string, you can indicate a range using the same format as the driver: `'0-2'` or `'0:2'`

Some repeated capabilities use a prefix before the number and this is optional

channels

nidigital.Session.**channels**

```
session.channels['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

pins

nidigital.Session.**pins**

```
session.pins['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

instruments

nidigital.Session.**instruments**

```
session.instruments['0-2'].channel_enabled = True
```

passes a string of `'0, 1, 2'` to the set attribute function.

pattern_opcode_events

nidigital.Session.**pattern_opcode_events**

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.pattern_opcode_events['0-2'].channel_enabled = True
```

passes a string of `'patternOpcodeEvent0, patternOpcodeEvent1, patternOpcodeEvent2'` to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.pattern_opcode_events['patternOpcodeEvent0-patternOpcodeEvent2'].
↪channel_enabled = True
```

passes a string of 'patternOpcodeEvent0, patternOpcodeEvent1, patternOpcodeEvent2' to the set attribute function.

conditional_jump_triggers

nidigital.Session.**conditional_jump_triggers**

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.conditional_jump_triggers['0-2'].channel_enabled = True
```

passes a string of 'conditionalJumpTrigger0, conditionalJumpTrigger1, conditionalJumpTrigger2' to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.conditional_jump_triggers['conditionalJumpTrigger0-
↪conditionalJumpTrigger2'].channel_enabled = True
```

passes a string of 'conditionalJumpTrigger0, conditionalJumpTrigger1, conditionalJumpTrigger2' to the set attribute function.

sites

nidigital.Session.**sites**

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.sites['0-2'].channel_enabled = True
```

passes a string of 'site0, site1, site2' to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.sites['site0-site2'].channel_enabled = True
```

passes a string of 'site0, site1, site2' to the set attribute function.

rio_events

nidigital.Session.**rio_events**

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.rio_events['0-2'].channel_enabled = True
```

passes a string of `'RIOEvent0, RIOEvent1, RIOEvent2'` to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.rio_events['RIOEvent0-RIOEvent2'].channel_enabled = True
```

passes a string of `'RIOEvent0, RIOEvent1, RIOEvent2'` to the set attribute function.

rio_triggers

nidigital.Session.**rio_triggers**

If no prefix is added to the items in the parameter, the correct prefix will be added when the driver function call is made.

```
session.rio_triggers['0-2'].channel_enabled = True
```

passes a string of `'RIOTrigger0, RIOTrigger1, RIOTrigger2'` to the set attribute function.

If an invalid repeated capability is passed to the driver, the driver will return an error.

You can also explicitly use the prefix as part of the parameter, but it must be the correct prefix for the specific repeated capability.

```
session.rio_triggers['RIOTrigger0-RIOTrigger2'].channel_enabled = True
```

passes a string of `'RIOTrigger0, RIOTrigger1, RIOTrigger2'` to the set attribute function.

Enums

Enums used in NI-Digital Pattern Driver

BitOrder

class nidigital.**BitOrder**

MSB

The most significant bit is first. The first bit is in the 2^n place, where n is the number of bits.

LSB

The least significant bit is first. The first bit is in the 2^0 place.

DigitalEdge

class nidigital.**DigitalEdge**

RISING

Asserts the trigger when the signal transitions from low level to high level.

FALLING

Asserts the trigger when the signal transitions from high level to low level.

DriveFormat

class nidigital.**DriveFormat**

NR

Drive format remains at logic level after each bit.

RL

Drive format returns to a logic level low after each bit.

RH

Drive format returns to a logic level high after each bit.

SBC

Drive format returns to the complement logic level of the bit after each bit.

FrequencyMeasurementMode

class nidigital.**FrequencyMeasurementMode**

BANKED

Frequency measurements are made serially for groups of channels associated with a single frequency counter for each group.

Maximum frequency measured: 200 MHz.

PARALLEL

Frequency measurements are made by multiple frequency counters in parallel.

Maximum frequency measured: 100 MHz.

HistoryRAMCyclesToAcquire

class nidigital.**HistoryRAMCyclesToAcquire**

FAILED

Acquires failed cycles.

ALL

Acquires all cycles.

HistoryRAMTriggerType

class nidigital.**HistoryRAMTriggerType**

FIRST_FAILURE

First Failure History RAM trigger

CYCLE_NUMBER

Cycle Number History RAM trigger.

PATTERN_LABEL

Pattern Label History RAM trigger

PPMUApertureTimeUnits

class nidigital.**PPMUApertureTimeUnits**

SECONDS

Unit in seconds.

PPMUCurrentLimitBehavior

class nidigital.**PPMUCurrentLimitBehavior**

REGULATE

Controls output current so that it does not exceed the current limit. Power continues to generate even if the current limit is reached.

PPMUMeasurementType

class nidigital.**PPMUMeasurementType**

CURRENT

The PPMU measures current.

VOLTAGE

The PPMU measures voltage.

PPMUOutputFunction

class nidigital.**PPMUOutputFunction**

VOLTAGE

The PPMU forces voltage to the DUT.

CURRENT

The PPMU forces current to the DUT.

PinState

class nidigital.**PinState**

ZERO

A digital state of 0.

ONE

A digital state of 1.

L

A digital state of L (low).

H

A digital state of H (high).

X

A digital state of X (non-drive state).

M

A digital state of M (midband).

V

A digital state of V (compare high or low, not midband; store results from capture functionality if configured).

D

A digital state of D (drive data from source functionality if configured).

E

A digital state of E (compare data from source functionality if configured).

NOT_A_PIN_STATE

Not a pin state is used for non-existent DUT cycles.

PIN_STATE_NOT_ACQUIRED

Pin state could not be acquired because none of the pins mapped to the instrument in a multi-instrument session had any failures.

SelectedFunction

class nidigital.**SelectedFunction**

DIGITAL

The pattern sequencer controls the specified pin(s). If a pattern is currently bursting, the pin immediately switches to bursting the pattern. This option disconnects the PPMU.

PPMU

The PPMU controls the specified pin(s) and connects the PPMU. The pin driver is in a non-drive state, and the active load is disabled. The PPMU does not start sourcing or measuring until Source or Measure(PpmuMeasurementType) is called.

OFF

Puts the digital driver in a non-drive state, disables the active load, disconnects the PPMU, and closes the I/O switch connecting the instrument channel.

DISCONNECT

The I/O switch connecting the instrument channel is open to the I/O connector. If the PPMU is sourcing, it is stopped prior to opening the I/O switch.

RIO

Yields control of the specified pin(s) to LabVIEW FPGA.

SequencerFlag

class nidigital.**SequencerFlag**

FLAG0

FLAG1

FLAG2

FLAG3

SequencerRegister

class nidigital.**SequencerRegister**

REGISTER0

REGISTER1

REGISTER2

REGISTER3

REGISTER4

REGISTER5

REGISTER6

REGISTER7

REGISTER8

REGISTER9

REGISTER10

REGISTER11

REGISTER12

REGISTER13

REGISTER14

REGISTER15

SoftwareTrigger

class nidigital.**SoftwareTrigger**

START

Overrides the start trigger.

CONDITIONAL_JUMP

Specifies to route a conditional jump trigger.

SourceDataMapping

class nidigital.**SourceDataMapping**

BROADCAST

Broadcasts the waveform you specify to all sites.

SITE_UNIQUE

Sources unique waveform data to each site.

TDREndpointTermination

class nidigital.**TDREndpointTermination**

OPEN

TDR channels are connected to an open circuit.

SHORT_TO_GROUND

TDR channels are connected to a short to ground.

TerminationMode

class nidigital.**TerminationMode**

ACTIVE_LOAD

The active load provides a constant current to a commutating voltage (Vcom).

VTERM

The pin driver drives Vterm.

HIGH_Z

The pin driver is in a non-drive state (in a high-impedance state) and the active load is disabled.

TimeSetEdgeType

class nidigital.**TimeSetEdgeType**

DRIVE_ON
Specifies the drive on edge of the time set.

DRIVE_DATA
Specifies the drive data edge of the time set.

DRIVE_RETURN
Specifies the drive return edge of the time set.

DRIVE_OFF
Specifies the drive off edge of the time set.

COMPARE_STROBE
Specifies the compare strobe of the time set.

DRIVE_DATA2
Specifies the drive data 2 edge of the time set.

DRIVE_RETURN2
Specifies the drive return 2 edge of the time set.

COMPARE_STROBE2
Specifies the compare strobe 2 of the time set.

TriggerType

class nidigital.**TriggerType**

NONE
Disables the start trigger.

DIGITAL_EDGE
Digital edge trigger.

SOFTWARE
Software start trigger.

WriteStaticPinState

class nidigital.**WriteStaticPinState**

ZERO
Specifies to drive low.

ONE
Specifies to drive high.

X
Specifies to not drive.

Exceptions and Warnings

Error

exception nidigital.errors.**Error**

Base exception type that all NI-Digital Pattern Driver exceptions derive from

DriverError

exception nidigital.errors.**DriverError**

An error originating from the NI-Digital Pattern Driver driver

UnsupportedConfigurationError

exception nidigital.errors.**UnsupportedConfigurationError**

An error due to using this module in an usupported platform.

DriverNotInstalledError

exception nidigital.errors.**DriverNotInstalledError**

An error due to using this module without the driver runtime installed.

DriverTooOldError

exception nidigital.errors.**DriverTooOldError**

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

DriverTooNewError

exception nidigital.errors.**DriverTooNewError**

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

InvalidRepeatedCapabilityError

exception nidigital.errors.**InvalidRepeatedCapabilityError**

An error due to an invalid character in a repeated capability

SelfTestError

exception nidigital.errors.**SelfTestError**

An error due to a failed self-test

RpcError

exception nidigital.errors.**RpcError**

An error specific to sessions to the NI gRPC Device Server

DriverWarning

exception nidigital.errors.**DriverWarning**

A warning originating from the NI-Digital Pattern Driver driver

Examples

You can download all nidigital examples for latest version here

nidigital_burst_with_start_trigger.py

Listing 1: (nidigital_burst_with_start_trigger.py)

```
#!/usr/bin/python

import argparse
import nidigital
import os
import sys


def example(resource_name, options, trigger_source=None, trigger_edge=None):

    with nidigital.Session(resource_name=resource_name, options=options) as session:

        dir = os.path.join(os.path.dirname(__file__))

        # Load the pin map (.pinmap) created using the Digital Pattern Editor
        pin_map_filename = os.path.join(dir, 'PinMap.pinmap')
        session.load_pin_map(pin_map_filename)

        # Load the specifications (.specs), levels (.digilevels), and timing (.
↪digitiming) sheets created using the Digital Pattern Editor
        spec_filename = os.path.join(dir, 'Specifications.specs')
        levels_filename = os.path.join(dir, 'PinLevels.digilevels')
        timing_filename = os.path.join(dir, 'Timing.digitiming')
        session.load_specifications_levels_and_timing(spec_filename, levels_filename,␣
↪timing_filename)
```

(continues on next page)

(continued from previous page)

```
24
25          # Apply the settings from the levels and timing sheets we just loaded to the␣
↪session
26          session.apply_levels_and_timing(levels_filename, timing_filename)
27
28          # Loading the pattern file (.digipat) created using the Digital Pattern Editor
29          pattern_filename = os.path.join(dir, 'Pattern.digipat')
30          session.load_pattern(pattern_filename)
31
32          if trigger_source is None:
33              print('Start bursting pattern')
34          else:
35              # Specify a source and edge for the external start trigger
36              session.start_trigger_type = nidigital.TriggerType.DIGITAL_EDGE
37              session.digital_edge_start_trigger_source = trigger_source
38              session.digital_edge_start_trigger_edge = nidigital.DigitalEdge.RISING if␣
↪trigger_edge == 'Rising' else nidigital.DigitalEdge.FALLING
39              print('Wait for start trigger and then start bursting pattern')
40
41          # If start trigger is configured, waiting for the trigger to start bursting and␣
↪then blocks until the pattern is done bursting
42          # Else just start bursting and block until the pattern is done bursting
43          session.burst_pattern(start_label='new_pattern')
44
45          # Disconnect all channels using programmable onboard switching
46          session.selected_function = nidigital.SelectedFunction.DISCONNECT
47      print('Done bursting pattern')
48
49
50  def _main(argsv):
51      parser = argparse.ArgumentParser(description='Demonstrates how to create and␣
↪configure a session that bursts a pattern on the digital pattern instrument using a␣
↪start trigger', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
52      parser.add_argument('-n', '--resource-name', default='PXI1Slot2,PXI1Slot3', help=
↪'Resource name of a NI digital pattern instrument. Ensure the resource name matches␣
↪the instrument name in the pinmap file.')
53      parser.add_argument('-s', '--simulate', default='True', choices=['True', 'False'],␣
↪help='Whether to run on simulated hardware or real hardware')
54      subparser = parser.add_subparsers(dest='command', help='Sub-command help')
55      start_trigger = subparser.add_parser('start-trigger', help='Configure start trigger')
56      start_trigger.add_argument('-ts', '--trigger-source', default='/PXI1Slot2/PXI_Trig0',
↪ help='Source terminal for the start trigger')
57      start_trigger.add_argument('-te', '--trigger-edge', default='Rising', choices=[
↪'Rising', 'Falling'], help='Trigger on rising edge or falling edge of start trigger')
58      args = parser.parse_args(argsv)
59
60      example(args.resource_name,
61              'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
62              args.trigger_source if args.command == 'start-trigger' else None,
63              args.trigger_edge if args.command == 'start-trigger' else None)
64
65
```

(continues on next page)

(continued from previous page)

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


def test_main():
    _main([])
    _main(['start-trigger'])


def test_example():
    resource_name = 'PXI1Slot2,PXI1Slot3'
    options = {'simulate': True, 'driver_setup': {'Model': '6571'}, }
    example(resource_name, options)

    trigger_source = '/PXI1Slot2/PXI_Trig0'
    trigger_edge = 'Rising'
    example(resource_name, options, trigger_source, trigger_edge)


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

nidigital_configure_time_set_and_voltage_levels.py

Listing 2: (nidigital_configure_time_set_and_voltage_levels.py)

```
#!/usr/bin/python

import argparse
import nidigital
import os
import sys


class VoltageLevelsAndTerminationConfig():
    def __init__(self, vil, vih, vol, voh, vterm, termination_mode, iol, ioh, vcom):
        self.vil = vil
        self.vih = vih
        self.vol = vol
        self.voh = voh
        self.vterm = vterm
        self.termination_mode = termination_mode
        self.iol = iol
        self.ioh = ioh
        self.vcom = vcom


class TimeSetConfig():
    def __init__(self, time_set_name, period, drive_format, drive_on, drive_data, drive_
↪return, drive_off, strobe_edge):
```

(continues on next page)

(continued from previous page)

```
        self.time_set_name = time_set_name
        self.period = period
        self.drive_format = drive_format
        self.drive_on = drive_on
        self.drive_data = drive_data
        self.drive_return = drive_return
        self.drive_off = drive_off
        self.strobe_edge = strobe_edge


def convert_drive_format(drive_format):
    converter = {'NR': nidigital.DriveFormat.NR,
                 'RL': nidigital.DriveFormat.RL,
                 'RH': nidigital.DriveFormat.RH,
                 'SBC': nidigital.DriveFormat.SBC}
    return converter.get(drive_format, None)


def example(resource_name,
            options,
            channels,
            voltage_config,
            time_set_config):

    with nidigital.Session(resource_name=resource_name, options=options) as session:

        dir = os.path.dirname(__file__)

        # Load pin map (.pinmap) created using Digital Pattern Editor
        pin_map_filename = os.path.join(dir, 'PinMap.pinmap')
        session.load_pin_map(pin_map_filename)

        # Configure voltage levels and terminal voltage through driver API
        session.channels[channels].configure_voltage_levels(voltage_config.vil, voltage_
↪config.vih, voltage_config.vol, voltage_config.voh, voltage_config.vterm)
        if voltage_config.termination_mode == 'High_Z':
            session.channels[channels].termination_mode = nidigital.TerminationMode.HIGH_
↪Z
        elif voltage_config.termination_mode == 'Active_Load':
            session.channels[channels].termination_mode = nidigital.TerminationMode.
↪ACTIVE_LOAD
            session.channels[channels].configure_active_load_levels(voltage_config.iol,␣
↪voltage_config.ioh, voltage_config.vcom)
        else:
            session.channels[channels].termination_mode = nidigital.TerminationMode.VTERM

        # Configure time set through driver API
        session.create_time_set(time_set_config.time_set_name)  # Must match time set␣
↪name in pattern file
        session.configure_time_set_period(time_set_config.time_set_name, time_set_config.
↪period)
        session.channels[channels].configure_time_set_drive_edges(time_set_config.time_
```

(continues on next page)

(continued from previous page)

```
↪set_name, convert_drive_format(time_set_config.drive_format),
                                                                time_set_config.drive_
↪on, time_set_config.drive_data,
                                                                time_set_config.drive_
↪return, time_set_config.drive_off)
        session.channels[channels].configure_time_set_compare_edges_strobe(time_set_
↪config.time_set_name, time_set_config.strobe_edge)

        # Load the pattern file (.digipat) created using Digital Pattern Editor
        pattern_filename = os.path.join(dir, 'Pattern.digipat')
        session.load_pattern(pattern_filename)

        # Burst pattern, blocks until the pattern is done bursting
        session.burst_pattern(start_label='new_pattern')
        print('Start bursting pattern')

        # Disconnect all channels using programmable onboard switching
        session.selected_function = nidigital.SelectedFunction.DISCONNECT
    print('Done bursting pattern')


def _main(argsv):
    parser = argparse.ArgumentParser(description='Demonstrates how to create an␣
↪instrument session, configure time set and voltage levels, and burst a pattern on the␣
↪digital pattern instrument.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2,PXI1Slot3', help=
↪'Resource name of a NI digital pattern instrument, ensure the resource name matches␣
↪the instrument name in the pinmap file.')
    parser.add_argument('-s', '--simulate', default='True', choices=['True', 'False'],␣
↪help='Whether to run on simulated hardware or on real hardware')
    parser.add_argument('-c', '--channels', default='PinGroup1', help='Channel(s)/Pin(s)␣
↪to configure')

    # Parameters to configure voltage
    parser.add_argument('--vil', default=0, type=float, help='The voltage that the␣
↪instrument will apply to the input of the DUT when the pin driver drives a logic low␣
↪(0)')
    parser.add_argument('--vih', default=3.3, type=float, help='The voltage that the␣
↪instrument will apply to the input of the DUT when the test instrument drives a logic␣
↪high (1)')
    parser.add_argument('--vol', default=1.6, type=float, help='The output voltage below␣
↪which the comparator on the pin driver interprets a logic low (L)')
    parser.add_argument('--voh', default=1.7, type=float, help='The output voltage above␣
↪which the comparator on the pin driver interprets a logic high (H)')
    parser.add_argument('--vterm', default=2, type=float, help='The termination voltage␣
↪the instrument applies during non-drive cycles when the termination mode is set to␣
↪Vterm')
    parser.add_argument('-term-mode', '--termination-mode', default='High_Z', choices=[
↪'High_Z', 'Active_Load', 'Three_Level_Drive'])
    parser.add_argument('--iol', default=0.002, type=float, help='The maximum current␣
↪that the DUT sinks while outputting a voltage below VCOM')
    parser.add_argument('--ioh', default=-0.002, type=float, help='The maximum current␣
```

(continues on next page)

(continued from previous page)

```
↪that the DUT sources while outputting a voltage above VCOM')
102     parser.add_argument('--vcom', default=0.0, type=float, help='The commutating voltage␣
↪level at which the active load circuit switches between sourcing current and sinking␣
↪current')
103
104     # Parameters to configure timeset
105     parser.add_argument('--period', default=0.00000002, type=float, help='Period in␣
↪second')
106     parser.add_argument('-format', '--drive-format', default='NR', choices=['NR', 'RL',
↪'RH', 'SBC'], help='Non-return | Return to low | Return to high | Surround by␣
↪complement')
107     parser.add_argument('--drive-on', default=0, type=float, help='The delay in seconds␣
↪from the beginning of the vector period for turning on the pin driver')
108     parser.add_argument('--drive-data', default=0, type=float, help='The delay in␣
↪seconds from the beginning of the vector period until the pattern data is driven to␣
↪the pattern value')
109     parser.add_argument('--drive-return', default=0.000000015, type=float, help='The␣
↪delay in seconds from the beginning of the vector period until the pin changes from␣
↪the pattern data to the return value, as specified in the format.')
110     parser.add_argument('--drive-off', default=0.00000002, type=float, help='The delay␣
↪in seconds from the beginning of the vector period to turn off the pin driver when the␣
↪next vector period uses a non-drive symbol (L, H, X, V, M, E).')
111     parser.add_argument('--strobe-edge', default=0.00000001, type=float, help='The time␣
↪in second when the comparison happens within a vector period')
112
113     args = parser.parse_args(argsv)
114     voltage_config = VoltageLevelsAndTerminationConfig(args.vil, args.vih, args.vol,␣
↪args.voh, args.vterm, args.termination_mode, args.iol, args.ioh, args.vcom)
115     time_set_config = TimeSetConfig("tset0", args.period, args.drive_format, args.drive_
↪on, args.drive_data, args.drive_return, args.drive_off, args.strobe_edge)
116     example(args.resource_name,
117             'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
118             args.channels,
119             voltage_config,
120             time_set_config)
121
122
123 def main():
124     _main(sys.argv[1:])
125
126
127 def test_main():
128     _main([])
129
130
131 def test_example():
132     resource_name = 'PXI1Slot2,PXI1Slot3'
133     options = {'simulate': True, 'driver_setup': {'Model': '6571'}, }
134     channels = 'PinGroup1'
135     voltage_config = VoltageLevelsAndTerminationConfig(vil=0, vih=3.3, vol=1.6, voh=1.7,␣
↪vterm=2,
136                                                        termination_mode='Active_Load',␣
```

(continues on next page)

(continued from previous page)

```
↪iol=0.002, ioh=-0.002, vcom=0)
    time_set_config = TimeSetConfig(time_set_name="tset0",
                                    period=0.00000002,
                                    drive_format='NR',
                                    drive_on=0, drive_data=0, drive_return=0.000000015,␣
↪drive_off=0.00000002, strobe_edge=0.00000001)
    example(resource_name, options, channels, voltage_config, time_set_config)


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

nidigital_ppmu_source_and_measure.py

Listing 3: (nidigital_ppmu_source_and_measure.py)

```
#!/usr/bin/python

import argparse
import nidigital
import os
import pytest
import sys
import time


def example(resource_name, options, channels, measure, aperture_time,
            source=None, settling_time=None, current_level_range=None, current_
↪level=None,
            voltage_limit_high=None, voltage_limit_low=None, current_limit_range=None,␣
↪voltage_level=None):

    with nidigital.Session(resource_name=resource_name, options=options) as session:

        dir = os.path.join(os.path.dirname(__file__))

        # Load pin map (.pinmap) created using Digital Pattern Editor
        pin_map_filename = os.path.join(dir, 'PinMap.pinmap')
        session.load_pin_map(pin_map_filename)

        # Configure the PPMU measurement aperture time
        session.channels[channels].ppmu_aperture_time = aperture_time
        session.channels[channels].ppmu_aperture_time_units = nidigital.
↪PPMUApertureTimeUnits.SECONDS

        # Configure and source
        if source == 'source-current':
            session.channels[channels].ppmu_output_function = nidigital.
↪PPMUOutputFunction.CURRENT

```

(continues on next page)

(continued from previous page)

```
            session.channels[channels].ppmu_current_level_range = current_level_range
            session.channels[channels].ppmu_current_level = current_level
            session.channels[channels].ppmu_voltage_limit_high = voltage_limit_high
            session.channels[channels].ppmu_voltage_limit_low = voltage_limit_low

            session.channels[channels].ppmu_source()

            # Settling time between sourcing and measuring
            time.sleep(settling_time)

        elif source == 'source-voltage':
            session.channels[channels].ppmu_output_function = nidigital.
↪PPMUOutputFunction.VOLTAGE

            session.channels[channels].ppmu_current_limit_range = current_limit_range
            session.channels[channels].ppmu_voltage_level = voltage_level

            session.channels[channels].ppmu_source()

            # Settling time between sourcing and measuring
            time.sleep(settling_time)

        pin_info = session.channels[channels].get_pin_results_pin_information()

        # Measure
        if measure == 'current':
            current_measurements = session.channels[channels].ppmu_measure(nidigital.
↪PPMUMeasurementType.CURRENT)

            print('{:<6} {:<20} {:<10}'.format('Site', 'Pin Name', 'Current'))

            for pin, current in zip(pin_info, current_measurements):
                print(f'{pin.site_number:<6d} {pin.pin_name:<20} {current:<10f}')
        else:
            voltage_measurements = session.channels[channels].ppmu_measure(nidigital.
↪PPMUMeasurementType.VOLTAGE)

            print('{:<6} {:<20} {:<10}'.format('Site', 'Pin Name', 'Voltage'))

            for pin, voltage in zip(pin_info, voltage_measurements):
                print(f'{pin.site_number:<6d} {pin.pin_name:<20} {voltage:<10f}')

        # Disconnect all channels using programmable onboard switching
        session.channels[channels].selected_function = nidigital.SelectedFunction.
↪DISCONNECT


def _main(argsv):
    parser = argparse.ArgumentParser(description='Demonstrates how to source/measure␣
↪voltage/current using the PPMU on selected channels/pins of the digital pattern␣
↪instrument',
                                     formatter_class=argparse.
```

(continues on next page)

(continued from previous page)

```
↪ArgumentDefaultsHelpFormatter)
    parser.add_argument('-n', '--resource-name', default='PXI1Slot2,PXI1Slot3', help=
↪'Resource name of a NI digital pattern instrument, ensure the resource name matches␣
↪the instrument name in the pinmap file.')
    parser.add_argument('-s', '--simulate', default='True', choices=['True', 'False'],␣
↪help='Whether to run on simulated hardware or on real hardware')
    parser.add_argument('-c', '--channels', default='DUTPin1, SystemPin1', help=
↪'Channel(s)/Pin(s) to use')
    parser.add_argument('-m', '--measure', default='voltage', choices=['voltage',
↪'current'], help='Measure voltage or measure current')
    parser.add_argument('-at', '--aperture-time', default=0.000004, type=float, help=
↪'Aperture time in seconds')
    subparser = parser.add_subparsers(dest='source', help='Sub-command help, by default␣
↪it measures voltage and does not source')

    source_current = subparser.add_parser('source-current', help='Source current')
    source_current.add_argument('-clr', '--current-level-range', default=0.000002,␣
↪type=float, help='Current level range in amps')
    source_current.add_argument('-cl', '--current-level', default=0.000002, type=float,␣
↪help='Current level in amps')
    source_current.add_argument('-vlh', '--voltage-limit-high', default=3.3, type=float,␣
↪help='Voltage limit high in volts')
    source_current.add_argument('-vll', '--voltage-limit-low', default=0, type=float,␣
↪help='Voltage limit low in volts')
    source_current.add_argument('-st', '--settling-time', default=0.01, type=float, help=
↪'Settling time in seconds')

    source_voltage = subparser.add_parser('source-voltage', help='Source voltage')
    source_voltage.add_argument('-clr', '--current-limit-range', default=0.000002,␣
↪type=float, help='Current limit range in amps')
    source_voltage.add_argument('-vl', '--voltage-level', default=3.3, type=float, help=
↪'Voltage level in volts')
    source_voltage.add_argument('-st', '--settling-time', default=0.01, type=float, help=
↪'Settling time in seconds')

    args = parser.parse_args(argsv)

    if args.source == 'source-current':
        example(
            args.resource_name,
            'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
            args.channels,
            args.measure,
            args.aperture_time,
            args.source,
            args.settling_time,
            args.current_level_range,
            args.current_level,
            args.voltage_limit_high,
            args.voltage_limit_low)
    elif args.source == 'source-voltage':
        example(
```

(continues on next page)

(continued from previous page)

```
            args.resource_name,
            'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
            args.channels,
            args.measure,
            args.aperture_time,
            args.source,
            args.settling_time,
            current_limit_range=args.current_limit_range,
            voltage_level=args.voltage_level)
    else:
        if args.measure == 'current':
            raise ValueError('Cannot measure current on a channel that is not sourcing␣
↪voltage or current')
        example(
            args.resource_name,
            'Simulate=1, DriverSetup=Model:6571' if args.simulate == 'True' else '',
            args.channels,
            args.measure,
            args.aperture_time)


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


def test_main():
    _main([])
    _main(['-m', 'voltage'])
    with pytest.raises(Exception):
        _main(['-m', 'current'])
    _main(['-m', 'voltage', 'source-current'])
    _main(['-m', 'current', 'source-current'])
    _main(['-m', 'voltage', 'source-voltage'])
    _main(['-m', 'current', 'source-voltage'])


def test_example():
    resource_name = 'PXI1Slot2,PXI1Slot3'
    options = {'simulate': True, 'driver_setup': {'Model': '6571'}, }
    channels = 'DUTPin1, SystemPin1'
    aperture_time = 0.000004

    example(resource_name, options, channels, 'voltage',
            aperture_time)
    with pytest.raises(Exception):
        example(resource_name, options, channels, 'current',
                aperture_time)

    settling_time = 0.01
    current_level_range = 0.000002
    current_level = 0.000002
    voltage_limit_high = 3.3
```

(continues on next page)

(continued from previous page)

```
    voltage_limit_low = 0
    example(resource_name, options, channels, 'voltage',
            aperture_time, 'source-current', settling_time,
            current_level_range, current_level,
            voltage_limit_high, voltage_limit_low)
    example(resource_name, options, channels, 'current',
            aperture_time, 'source-current', settling_time,
            current_level_range, current_level,
            voltage_limit_high, voltage_limit_low)

    current_limit_range = 0.000002
    voltage_level = 3.3
    example(resource_name, options, channels, 'voltage',
            aperture_time, 'source-voltage', settling_time,
            current_limit_range=current_limit_range,
            voltage_level=voltage_level)
    example(resource_name, options, channels, 'current',
            aperture_time, 'source-voltage', settling_time,
            current_limit_range=current_limit_range,
            voltage_level=voltage_level)


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

gRPC Support

Support for using NI-Digital Pattern Driver over gRPC

SessionInitializationBehavior

class nidigital.**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 nidigital.**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** (*nidigital.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

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z